Universidade do Minho
Escola de Engenharia

Filipe André Lopes Salgado

Bases de Dados em Grafos: Estudo Exploratório no Âmbito das Bases de Dados *NoSQL*

Dissertação de Mestrado

Mestrado integrado em Engenharia e Gestão de Sistemas de Informação

Trabalho efetuado sob a orientação de

Professor Doutor José Luís Mota Pereira

Outubro de 2017

DECLARAÇÃO

Nome: Filipe André Lopes Salgado

Endereço eletrónico: a69183@alunos.uminho.pt

Telefone: 918 455 360

Cartão do Cidadão: 14610971

Título da dissertação: Bases de Dados em Grafos: Estudo Exploratório no Âmbito das Bases de Dados *NoSQL*

Orientador:

Professor Doutor José Luís Mota Pereira

Ano de conclusão: 2017

Mestrado integrado em Engenharia e Gestão de Sistemas de Informação.

Universidade do Minho, 30/10/2017

Assinatura:

AGRADECIMENTOS

Obrigado acima de tudo aos meus pais que me ajudaram e suportaram para que eu conseguisse chegar até aqui. Sem eles, nada disto seria possível. Obrigado por tudo, pela educação que me deram, pelos valores transmitidos, por todo o dinheiro investido na minha formação e por me apoiarem em tudo. São sem dúvida a minha fonte de inspiração! Aos meus irmãos, por me terem ajudado em tudo o quanto lhes era possível e impossível.

Obrigado ao meu orientador, professor Dr. José Luís Mota Pereira, por me ter ajudado em tudo quanto pôde, e por me ter transmitido o conhecimento sobre este fascinante mundo dos sistemas de informação desde a primeira unidade curricular do curso, a de "Tópicos de Bases de Dados". Um obrigado também a todos os meus professores que estiveram comigo durante o percurso universitário.

Aos meus amigos de Guimarães e da universidade por toda a paciência que perderam comigo e por todas as horas que despenderam comigo para que conseguisse completar estes cinco anos.

A todos os que de uma maneira ou de outra me ajudaram a que tudo isto fosse possível, o meu muito obrigado!

RESUMO

Depois de várias décadas de grande sucesso e bons serviços prestados às organizações, a tecnologia relacional de bases de dados tem vindo a ser desafiada por uma nova classe de tecnologias de bases de dados, a que se deu a designação genérica de *NoSQL* (*Not only SQL*). Para este facto contribuíram decisivamente os recentes desenvolvimentos na área a que se tem vindo a chamar *Big Data*, na qual, dada a complexidade e volume dos conjuntos de dados a gerir, o tradicional modelo relacional começou a apresentar dificuldades.

Dentro desta nova classe de tecnologias de bases de dados surgiram diferentes propostas, com distintas proveniências e áreas de aplicação, vulgarmente classificadas em quatro grupos, de acordo com o seu modelo de dados: orientado a colunas, orientado a documentos, pares Chave-Valor e orientado a grafos. Dada a grande diversidade de propostas atualmente existentes em cada um dos modelos de bases de dados *NoSQL*, torna-se pertinente compreender as suas características específicas e áreas de aplicação mais relevantes, enquanto, simultaneamente se vincam as suas diferenças relativamente às tradicionais bases de dados relacionais.

Em particular, aborda-se uma das quatro classes de bases de dados em que o mundo *NoSQL* se divide – as bases de dados orientadas a Grafos, de que produtos como o *Neo4J*, *OrientDB* ou *Titan* são alguns dos principais representantes. Neste trabalho é realizado uma caracterização do contexto *Big Data* e analisado o papel das bases de dados *NoSQL* nesse mesmo contexto. De seguida, são estudados os produtos *NoSQL* atualmente mais representativos de entre as bases de dados em grafos tais como os referidos anteriormente, e é feito uma análise comparativa entre um deles – *Neo4J*, e uma base de dados relacional – *SQL Server 2014*. São identificadas vantagens e desvantagens desse produto relativamente à base de dados relacional, assim como é realizado uma comparação de desempenho de diversas consultas entre o *SQL Server 2014* e o *Neo4J* em 4 cenários distintos e com diferente número de acessos simultâneos, onde, de uma maneira geral, a base de dados em Grafo desempenha melhor.

Palavras-Chave: *Big Data, NoSQL, SQL, Cypher, Graph Databases, Relational Databases*

ABSTRACT

After several decades of great success and good services to organizations, relational databases technology has been challenged by a new class of database technologies, NoSQL (Not only SQL) Databases. Recent developments in Big Data, where there are datasets with a lot of complexity and volume, the traditional relational model has begun to present difficulties to manage them.

Within this new class of databases technologies, different proposals have emerged, with different origins and areas of application. These areas of application are usually classified into four groups, according to their data model: column oriented databases, document oriented databases, Key-Value databases and graph oriented databases.

Once there is a wide range of NoSQL Database proposals currently available, it is relevant to understand their specific characteristics and most relevant applications areas, and also, to find the differences between these 4 models comparing to the relational databases.

In this project, we focus in one of four NoSQL databases - the Graph-oriented Databases, where products like Neo4J, OrientDB or Titan are the main representatives. Thus, with this work we characterize the Big Data context and analyze the role of the NoSQL Databases in this same context. Then, are studied the most representative NoSQL products of Graph Databases such as those mentioned above, and made a comparative analysis between them. It is also identified advantages and disadvantages of one of these products - Neo4J comparing to one relational database - SQL Server 2014. Also, it is compared the performance of the products mentioned above, in four distinct scenarios with different number of simultaneous access, where, in a general way, Neo4J performs better.

KEYWORDS: BIG DATA, NOSQL, SQL, CYPHER, GRAPH DATABASES, RELATIONAL DATABASES

ÍNDICE

LISTA DE FIGURAS

Lista de Tabelas

LISTA DE ABREVIATURAS, SIGLAS E ACRÓNIMOS

ACID - Atomicity, Consistency, Isolation, Durability.

AP - Availability e *Partition Tolerance.*

API - Application Programming Interface.

BASE - Basic Availability, Soft-state, Eventual-consistency.

BSON - Binary JavaScript Object Notation.

CA - Consistency e *Availability.*

CAP - Consistency, Available and Partition Tolerance.

CP - Consistency e *Partition Tolerance.*

CRUD - Create, Read, Update, Delete.

CSV - Comma Separated-Values.

DSL - Domain-Specific Language.

GQL - Google Query Language.

HTML - Hypertext Markup Language.

HTTP - Hypertext Transfer Protocol.

*IT/TI - Information Technologies/*Tecnologias da Informação.

JDB - Java Debug.

JSON - JavaScript Object Notation.

Mb - Megabytes.

NoSQL - Not only SQL.

RDF - Resource Description Framework.

REST - Representational State Transfer.

ROI - Return on investment.

SGBD - Sistema de Gestão de Bases de Dados.

SGBDR - Sistema de Gestão de Bases de Dados Relacionais.

SQL - Structured Query Language.

URL - Uniform Resource Locator.

XML - Extensible Markup Language.

1. Introdução e Desenvolvimento do Trabalho Proposto

Neste primeiro capítulo começaremos por fazer um enquadramento relativamente ao termo *Big Data*, onde é caracterizado o papel das bases de dados *NoSQL* nesse contexto, especialmente o das bases de dados em grafos. Seguidamente, são apresentados os objetivos para esta dissertação, assim como qual a estrutura deste documento.

1.1 Enquadramento

Todos os dias, grandes quantidades de novos dados surgem na *Web*, num fenómeno que vem a ser denominado de *Big Data*. Estes dados são tipicamente complexos, de grande volume e sem qualquer tipo de estrutura associada. Assim, os SGBDR começaram a apresentar grandes dificuldades no seu armazenamento e análise. Com estes problemas, diversos produtos de armazenamento de dados têm vindo a surgir, as chamadas bases de dados *NoSQL*. Estas, dividem-se tipicamente em quatro grandes grupos: orientadas a Colunas, orientadas a Documentos, pares Chave-Valor e orientadas a Grafos.

Especificamente as bases de dados em grafos, são ainda pouco exploradas, contudo apresentam um potencial enorme derivado do facto de permitirem representar perfeitamente relações complexas entre dados. Estas relações são dificilmente representadas nos outros grupos de bases de dados *NoSQL*, isto é, nas bases de dados em Colunas, Chave-Valor e orientada a Documentos. A informação é dos recursos mais valiosos que atualmente existem, e se estiver interligada, mais valioso é. Assim, pretende-se estudar as bases de dados em grafos, e verificar em que níveis estas conseguem superar o modelo relacional de bases de dados.

1.2 Objetivos

Com este trabalho pretende-se fazer uma caracterização do contexto *Big Data* e analisar o papel das bases de dados *NoSQL* nesse mesmo contexto. Em particular, pretende-se:

1) estudar detalhadamente alguns produtos *NoSQL* mais representativos de entre as bases de dados em grafos e fazer uma análise comparativa entre esses diversos produtos;

2) identificar vantagens e desvantagens desses produtos relativamente às bases de dados relacionais;

3) comparar o desempenho do produto *NoSQL* representativo das bases de dados em grafos e das bases de dados relacionais, sendo este o grande resultado final.

Deste modo responde-se à seguinte questão de investigação: "Qual o papel das bases de dados em grafos no contexto de suporte ao *Big Data* proporcionado pelas bases de dados *NoSQL*?".

1.3 Estrutura do Documento

Este documento encontra-se estruturado em seis capítulos organizados pela seguinte forma:

- **Capítulo 1 - Introdução e Enquadramento do Trabalho Proposto –** Neste primeiro capítulo é feita uma introdução sobre o tema *Big Data*, assim como é realizado um enquadramento com a situação atual, onde podemos verificar porque é que este tema é tão importante nos dias de hoje. São também referidos os objetivos e os resultados propostos para esta dissertação.

- **Capítulo 2 – Abordagem Metodológica –** Neste capitulo é especificado qual a abordagem metodológica utilizada neste projeto de dissertação assim como a explicação da estratégia utilizada para a revisão de literatura.

- **Capítulo 3 - Revisão de Literatura –** Neste capítulo são abordadas várias temáticas assim como toda a teoria relevante para a execução deste projeto. Começaremos por explicar qual a origem do termo *Big Data* e o que isso significa para diversos autores conhecidos. Como do *Big Data* surgiram as bases de dados *NoSQL*, é realizada também uma revisão de literatura nesta temática onde são explicados diversos conceitos. Abordamos também o Teorema *CAP* no âmbito das bases de dados *NoSQL*, e consecutivamente as transações *BASE* e *ACID*, com alguns exemplos de bases de dados que usam essas transações. Abordaremos os 4 tipos de bases de dados *NoSQL*, com mais ênfase e detalhe para as bases de dados em grafos, onde iremos comparar três dos produtos mais representativos desta gama de bases de dados. Por fim, é realizada uma comparação entre a tecnologia relacional de bases de dados e a tecnologia *NoSQL* através de diferentes consultas a diferentes cenários que variam na quantidade de dados. São também apresentadas algumas vantagens, desvantagens e limitações da tecnologia *NoSQL*.

- **Capítulo 4 - Recolha, Análise, Tratamento e Carregamento dos Dados –** Neste capítulo é descrito o processo desde a recolha dos dados, passando pela sua análise e tratamento até ao seu carregamento através dos diferentes cenários nas respetivas bases de dados relacionais e bases de dados em grafos.

- **Capítulo 5 - *Benchmarking* –** Neste capitulo é descrito o processo de *Benchmarking* às duas bases de dados presentes nesta dissertação, assim como uma análise aos resultados obtidos.

- **Capítulo 6 - Conclusão –** Neste último capítulo é realizada uma conclusão deste projeto de dissertação.

2. Abordagem Metodológica

Para a realização deste projeto, a abordagem metodológica seguida foi a *Design Science Research* (DSR) que é um conjunto analítico de técnicas e de perspetivas (complementa as perspetivas positivistas e interpretativistas) para a realização de estudos em sistemas de informação (Vaishnavi & Kuechler, 2004). Envolve a análise do uso e o desempenho dos artefactos projetados para compreender, explicar e melhorar o comportamento dos aspetos em estudo (Vaishnavi & Kuechler, 2004). Esta metodologia também é referida a um conjunto de orientações e métodos específicos para o processo de criação, construção e validação de um artefacto no contexto de inovação em TI (Wang & Wang, 2013). Estes artefactos são normalmente concebidos para satisfazer uma necessidade ou para atender a um objetivo (Simon, 1997). Segundo o mesmo autor, esta metodologia resume-se nos seguintes passos:

1. "Conscientização";
2. "Sugestão" para resolver esse problema;
3. "Desenvolvimento" da sugestão proposta para o problema;
4. "Avaliação" de modo a verificar a qualidade do artefacto criado para solucionar o problema;
5. "Conclusão".

Já para outro autor, (Peffers et al., 2008), a DSR encontra-se dividida em 6 fases, como se pode observar na figura 1.

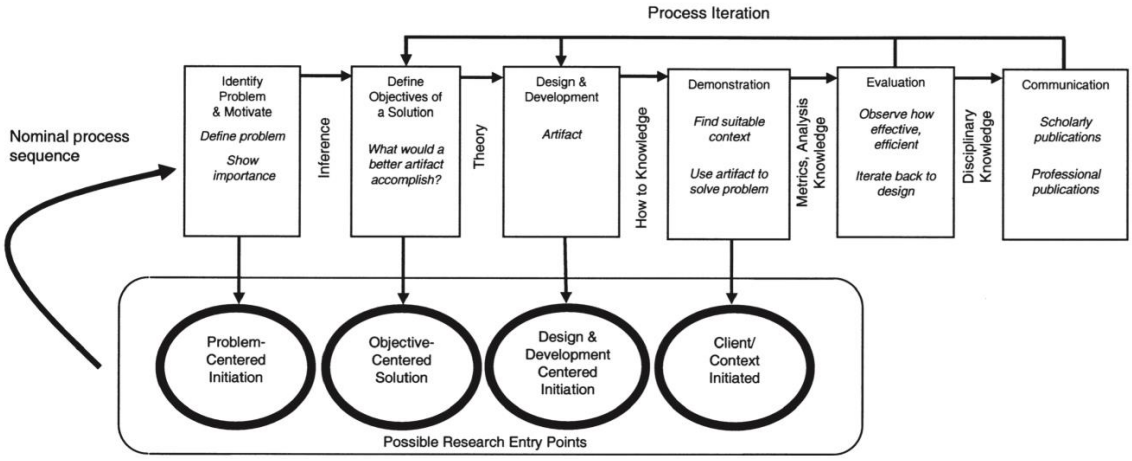

Figura 1 - Design Science Research para os Sistemas de Informação. Retirado de (Peffers et al., 2008).

Todo o processo da dissertação, incluiu estes 6 passos:

1. Identificar o problema e motivar: Neste ponto o principal objetivo passa por definir o problema e demonstrar a importância que este tem para toda a comunidade académica, cientifica e profissional. Esta identificação do problema é bastante importante para providenciar uma solução, pelo que a sua descrição convém ser o mais clara possível, de modo a que a solução consiga obter e captar a complexidade do mesmo. O investigador e todas as partes interessadas têm de ser capazes de entender bem o problema assim como dar valor a uma possível solução;
2. Definir objetivos da solução: Através da definição do problema, este passo foca-se na definição dos objetivos para a solução do problema tendo em conta todo o conhecimento sobre o problema e sobre possíveis soluções, podendo estes ser qualitativos ou quantitativos;
3. Conceção e desenvolvimento: Criar o artefacto que resulta da investigação. Isto inclui definir os aspetos e/ou funcionalidades que a solução iria ter, ou seja, a sua arquitetura;
4. Demonstração: Este passo tem como objetivo demonstrar o artefacto criado, depois das diversas experiências e simulações a este submetidas;
5. Avaliação: Uma vez demonstrado o artefacto, torna-se necessário fazer uma avaliação ao artefacto, ou seja, verificar qual o desempenho dele para a solução dele e se os objetivos traçados para o artefacto foram atingidos;
6. Comunicação: Uma vez avaliado o artefacto, este tem de ser comunicado e divulgado a investigadores, profissionais da área, e todos os restantes interessados na solução que o artefacto traz para toda a comunidades. Esta comunicação pode ser realizada através da divulgação em conferências, da publicação de artigos em revistas cientificas ou, através da publicação de uma dissertação, da qual este documento é um exemplo.

Durante este projeto seguiremos uma abordagem exploratória, uma vez que nesta área ainda há pouco conhecimento e informação disponível. Pode-se definir Estudo exploratório como um "estudo preliminar designado para desenvolver ou refinar hipóteses, ou testar e definir métodos de coleção de dados" (Richardson, 1999). Estes estudos exploratórios servem, para perante fenómenos desconhecidos, aumentar o grau de familiaridade, obtendo assim informações sobre a possibilidade de efetuar uma investigação mais completa sobre determinado contexto e estabelecer prioridades para investigações futuras entre várias outras utilizações (Sampieri et al., 2006).

Esta metodologia difere-se dos estudos descritivos ou explicativos, uma vez que é mais ampla e diversa que os dois últimos (Sartori & Révillion, 2001).
Sendo assim, irão ser exploradas as diversas soluções existentes e optar pela melhor.

2.1 Estratégia da Revisão de Literatura

A principal estratégia de revisão de literatura passou por recolher informação nos principais repositórios académicos e científicos dos quais se destacam os seguintes:

- *Google Scholar*;
- *Scopus*;
- *Microsoft Academic Research*;
- *Springer*;
- *B-On*;
- *Web of Knowledge*;
- *RepositoriUM*;

A seleção do material dos repositórios académicos e científicos passou por algumas preferências, nomeadamente:

- Preferência por artigos mais recentes (posteriores a 2010);
- Relevância do/s autor/es no tema;
- Maior número de citações dos artigos científicos;
- Tipo de documento: preferência por Livros, depois Artigos, Dissertação, *White paper* e só depois *blogs* profissionais;
- Preferência a artigos escritos na língua inglesa;

A recolha da literatura associada foi realizada durante os meses de julho de 2016 a fevereiro de 2017. Durante os meses seguintes foram realizadas algumas recolhas de literatura, de modo a colmatar algumas falhas na informação recolhida anteriormente.

A recolha da literatura associada foi fundamentada em pesquisas de palavras chaves nos repositórios académicos e científicos referidos anteriormente. Algumas das palavras ou menções utilizadas foram:

- *Big Data*;
- *NoSQL Databases*;
- *CAP Theorem*;
- *BASE* vs *ACID*;
- *Graph Databases*;
- *Key/Value Databases*;
- *Document Databases*;
- *Column Databases*;
- *Relational Databases*;
- *NoSQL* vs *SQL Databases*;
- *Neo4J*;
- *OrientDB*;
- *Titan*;
- *Graph Databases modelation*;
- *Graph databases vs relational databases*;
- *Cypher*;
- *Gremlin*;

3. Revisão de Literatura

3.1 Big Data

O termo *Big Data* está desde logo associado a um enorme volume de dados. Desde a invenção dos computadores, grandes quantidades de dados têm sido geradas a uma taxa de crescimento cada vez mais elevada (Yaqoob et al., 2016). Se as evoluções em dispositivos móveis, sensores digitais, comunicações e computação têm vindo a gerar enormes volumes de dados, o processo ao nível das tecnologias de armazenamento têm proporcionado meios para armazenar grandes quantidades de dados (Bryant, Katz, & Lazowska, 2008). Estima-se que o volume de dados na Internet cresça 40% por ano e 50 vezes até 2020 (Waal-Montgomery, 2016). O mesmo autor (Waal-Montgomery, 2016), refere que irá haver uma procura muito grande em conhecimentos de *Big Data*, referindo o caso do Reino Unido onde se estima um aumento de 160%. Outro autor, refere que devido a este grande crescimento, a produção de dados em 2020 irá ser 44 vezes superior ao que ocorreu em 2009 (Khan et al., 2014). Segundo (Reckoning, 2014), o total de dados criados e partilhados no mundo aumentou nove vezes em apenas 5 anos (de quase zero até dois *zettabytes* no fim do ano de 2011), contudo o crescimento não se fica por aqui, pois até 2015, a criação de dados quase quadruplicou. Podemos ver essa informação na figura 2.

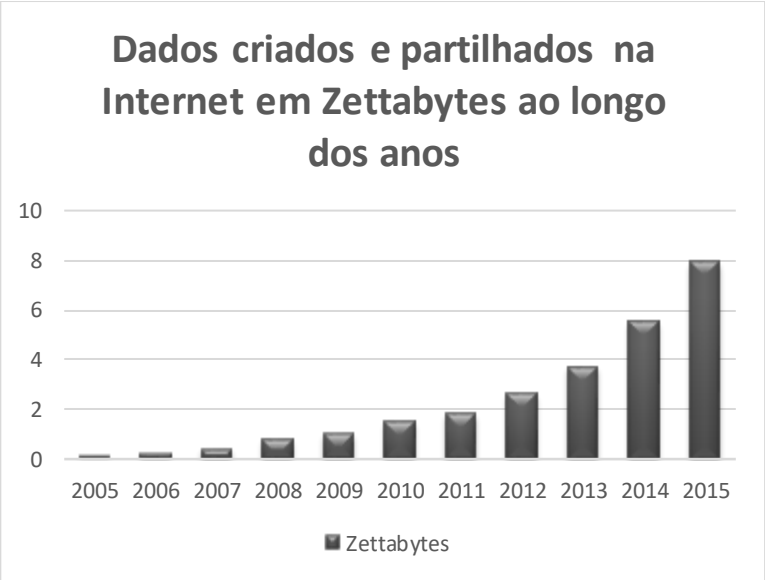

Figura 2 - Dados criados e partilhados na Internet. Adaptado de (Reckoning, 2014).

Como podemos observar, os dados e o mundo digital estão a crescer a uma taxa bastante elevada. Contudo estes dados são complexos pois, além de grandes quantidades, uma grande parte deles não tem uma estrutura fixa. Assim, vários autores defendem diferentes perspetivas sobre o termo *Big Data*. Segundo (Brown et al., 2011), *Big Data* é um conjunto de dados, cujo tamanho vai para além da capacidade das ferramentas típicas de bases de dados no que respeita à captura, armazenamento, gestão e análise de dados. Para (Dumbill, 2012) *Big Data* são os dados que excedem a capacidade de processamento dos sistemas convencionais de bases de dados. Outro autor (Marr, 2015) defende que *Big Data* está ligado à capacidade que as organizações têm em recolher e analisar o vasto volume de dados que é gerado no mundo. Por sua vez, (Moniruzzaman & Hossain, 2013) definem *Big Data* como uma gigante coleção de dados, que devido ao seu tamanho e crescimento constante, não pode ser gerido com a tecnologia relacional de base de dados. Para (Garlasu et al., 2013), o termo *Big Data* é um termo recente e originário da necessidade de grandes companhias como o *Yahoo*, *Google* e *Facebook,* para gerir grandes quantidades de dados. Foi (Laney, 2001), quem associou ao *Big Data* as suas principais características, os "3 V's", que, segundo o autor são o Volume, Velocidade e Variedade. (Chen, P., & Zhang, 2014) definem Volume como o tamanho dos dados, Velocidade como o tempo entre a entrada e saída dos dados, e Variedade como a fonte e os tipos de dados. A *IBM* e a *Microsoft* adicionam ainda a Veracidade ou Variabilidade a estes 3 V's.

Citando (Gantz & Reinsel, 2011), "As Tecnologias de *Big Data* descrevem uma nova geração de tecnologias e arquiteturas, desenhadas para extrair valor económico de grandes volumes de dados, de uma grande variedade de dados, ao permitir alta velocidade de captura de dados, de descoberta e/ou de análise", este autor caracteriza *Big Data* como 4 V's, sendo eles o Valor, Volume, Variedade e Velocidade. Por seu lado, há autores que caracterizam *Big Data* segundo 5 V's. (Marr, 2015) defende a inclusão do Valor e Veracidade como características do *Big Data*. Bernard Marr defende que uma grande quantidade de dados é útil quando estes nos trazem Valor, pelo que podemos argumentar que o Valor é definitivamente um "V" dos mais importantes de *Big Data*. Segundo *Bernard Marr*, a Veracidade refere-se à confusão e confiabilidade dos dados. Com grandes quantidades de dados, a qualidade e a precisão são menos controláveis, mas hoje em dia as tecnologias de *Big Data* e de análise de dados permitem trabalhar com este tipo de dados. *Bernard Marr* refere ainda que "Os volumes de dados muitas vezes compensam a falta de qualidade ou precisão".

Sintetizando, alguns autores caracterizam *Big Data* segundo 3, 4 ou 5V's, mas todos eles referem que este é um termo utilizado para descrever o crescimento exponencial de dados, que aparece como um conjunto de tecnologias, processos e práticas que permitem às organizações retirar partido da análise de grandes volumes de dados para a tomada de decisão. Estas informações fornecidas para os responsáveis da tomada de decisão são defendidas por (Stidston, M., 2014), como os "5R's do Big Data", e não os "5 V's":

- ***Relevant***: A informação fornecida tem de ser relevante e útil para o resultado do negócio. Essa informação é difícil de ser filtrada pois existe muito volume e variedade nos dados;
- ***Real time***: Existir acesso aos dados em tempo real sempre que necessário;
- ***Realistic***: Todos os dados extraídos devem ser realísticos e devem ir de encontro ao negócio em questão. Está relacionado com a perceção dos dados;
- ***Reliable***: A qualidade dos dados é fundamental para a confiabilidade e eficácia dos conjuntos de resultados. Está relacionado com a qualidade dos dados;
- ***ROI***: O investimento no *Big Data* tem de trazer retorno ao decisor. A gestão eficaz e a análise dos dados permitem melhores decisões no negócio, maximizando assim o Retorno do Investimento (*ROI*) do seu sistema de processamento de dados.

Sendo assim, podemos definir *Big Data* como sendo a gestão de grandes volumes de dados. Esses dados têm diversas características que torna difícil o seu tratamento e análise, mas que são contornadas por um conjunto de ferramentas adequadas para serem manipulados.
Com este novo contexto, e depois de várias décadas de grande sucesso e bons serviços prestados às organizações, a tecnologia relacional de base de dados tem vindo a ser desafiada por uma nova geração de tecnologias de bases de dados, a que se dá o nome de *NoSQL* (*Not only SQL*).

3.2 Surgimento do *NoSQL*

O mundo digital está a crescer a olhos vistos, tornando-se cada vez mais complexo. Como referido no ponto 3.1, o volume de dados está a aumentar a uma taxa muito elevada e a variedade dos dados é cada vez maior. Graças a este fenómeno, popularmente chamado de *Big Data*, surgiram novas tecnologias de bases de dados, as *NoSQL* (*Not Only SQL*), que dispõem de melhorias ao nível do

armazenamento e processamento de grandes quantidades de dados (Vieira et al., 2012), visando assim combater a dificuldade que a tecnologia relacional de base de dados tem em gerir esta nova geração de dados. Antes de percebermos o que é uma base de dados *NoSQL*, vamos tentar perceber o que é uma base de dados. Base de dados é um conjunto de dados com algumas características básicas, tais como o armazenamento (sistema para guardar dados), a recuperação (possibilidade de o sistema recuperar os dados guardados) e a visualização. Pode ter outras características opcionais, tais como a possibilidade de compartilhamento, segurança, consistência, etc. A tecnologia de base de dados atualmente mais conhecida é a tecnologia relacional, que foi proposta por E. F. Codd em 1970 e, "na altura foi uma grande inovação, uma vez que veio simplificar o armazenamento de dados e o seu processamento, ao migrar os dados para tabelas relacionadas entre si por campos comuns, manipulados por uma linguagem de alto-nível – a *SQL*" (Sousa, 2015).

A primeira referência a uma base de dados que não utilizava uma interface *SQL* foi introduzida em 1980 por *Carlo Strozzi*, que preferiu referir *NoSQL* como "*noseequel*" ou "*NoRel*", que era a principal diferença entre essa tecnologia e as que já existiam (Lith & Mattsson, 2010). Contudo, a origem do *NoSQL* pode estar relacionada com o modelo desenvolvido pela *Google*, o *BigTable* (Chang et al., 2006). O termo *NoSQL* começou a ganhar popularidade no inicio de 2009 (Vicknair et al., 2010), muito devido aos requisitos de mudança impostas ao mundo do *IT*, causados principalmente pelas redes sociais, e pelos serviços de *cloud*. Alguns tipos de bases de dados *NoSQL* têm emergido e ganho notoriedade.

A necessidade de distribuir dados por múltiplos servidores, de armazenar dados estruturados e não estruturados, a necessidade de fazer consultas rápidas que envolvessem muitos "*joins*", e em que não houvesse perda de dados levou ao desenvolvimento de muitas bases de dados *NoSQL*. A crescente popularização das redes sociais e a geração de conteúdo por dispositivos móveis, bem como o número cada vez maior de pessoas e dispositivos conectados, fez com que houvesse alguma dificuldade em armazenar esses dados. Apesar da tecnologia relacional ser muito escalável, quanto maior for o seu tamanho, mais dispendiosa será. A tecnologia relacional segue o princípio *ACID* (*Atomicity*, *Consistency*, *Isolation* e *Durability*) (Leavitt, 2010), e são baseadas em transações, sendo que estas asseguram consistência em todas as situações de gestão de dados (Moniruzzaman & Hossain, 2013). Este princípio é rejeitado pelas bases de dados *NoSQL*, pelo que a maioria destes produtos seguem o princípio *CAP*, que será explicado num ponto mais adiante.

As bases de dados *NoSQL* são indicadas para aplicações que irão trabalhar com uma quantidade enorme de dados, que têm exigências ao nível da velocidade nas suas consultas e escritas em grandes volumes de dados.

As bases de dados *NoSQL* são tipicamente divididas em 4 tipos. Segundo (Tudorica & Bucur, 2011), existem 4 tipos de bases de dados *NoSQL*:

- Bases de dados orientadas a Documentos;
- Bases de dados Chave-Valor;
- Bases de dados orientadas a Grafos;
- Bases de dados orientadas a Colunas.

Cada um deste tipo de bases de dados contém um vasto leque de características que vão ser estudados num ponto mais à frente.

Na figura 3, podemos observar uma figura-resumo, onde estão presentes vários produtos identificativos de cada tipo de base de dados, de entre as relacionais e não relacionais, onde se incluem algumas *NoSQL*.

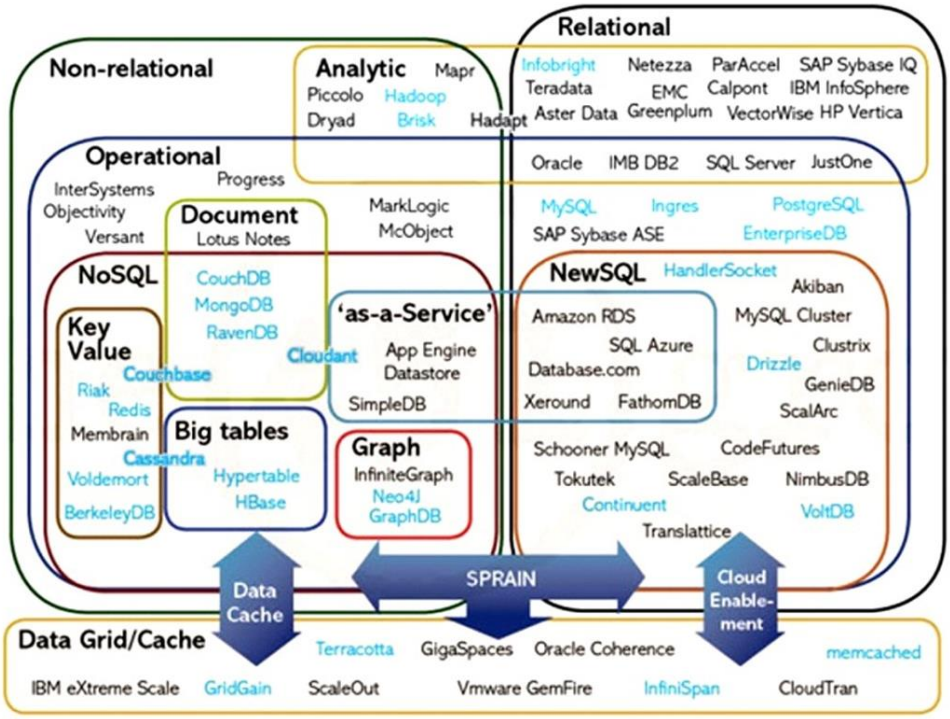

Figura 3 - Produtos representativos de diversos tipos de bases de dados. Retirado de (Moniruzzaman & Hossain, 2013).

3.2.1 Teorema *CAP*

O Teorema *CAP* (*Consistency*, *Availability* e *Partition Tolerance*) foi proposto por *Brewer* no ano de 2000 durante o "*ACM Symposium on Principles of Distributed Computing*", e comprovado por *Seth Gilbert* e *Nancy Lynch* no ano de 2002. Este teorema é constituído por três principais requisitos. Estes três requisitos são descritos por (Sousa, 2010) como:

- ***Consistency*:** É a característica que descreve como e se o sistema fica consistente após alguma operação. Isto significa que, se uma vez escrito um registo, este fica disponível podendo assim ser utilizado imediatamente. O sistema pode ser classificado em fortemente consistente (Sistemas com características *ACID*) ou eventualmente consistente (Sistemas com características *BASE*). Todos os nós de um sistema têm acesso à mesma versão de dados;

- ***Availability*:** É a característica que assegura que um sistema está disponível durante um período de tempo. O sistema pode ser classificado, num determinado período de tempo, como estar ou não estar disponível. O sistema está disponível quando todos os clientes conseguem aceder a pelo menos uma cópia dos dados, mesmo que um nó esteja desligado da rede;

- ***Partition Tolerance*:** É a característica que permite ao sistema, apesar de sofrer uma falha de rede, continuar a operar. Se uma base de dados se encontrar particionada e a ligação entre duas delas falhar, o sistema é tolerante a falhas se ainda fornecer operações de escrita e/ou leitura enquanto particionada. Se não fornecer essas operações, o sistema não é tolerante à partição.

No seu conhecido Teorema *CAP*, (Brewer, 2000) sugere que as bases de dados apenas podem funcionar bem em dois dos três requisitos descritos anteriormente, como por exemplo ter consistência e disponibilidade, mas no entanto não ser tolerante à partição. A figura 4 demonstra todas as possíveis combinações:

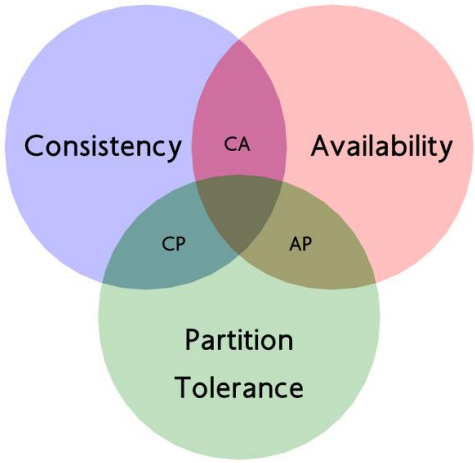

Figura 4 - Teorema CAP. Retirado de (Viraj, T., 2016).

Como podemos observar na figura 4, existem três tipos de combinações possíveis dentro do teorema *CAP*. As combinações são as seguintes:

- **CA:** Este grupo de sistemas fornece um serviço consistente e disponível, contudo não tolera partições. Pode tornar-se inconsistente no caso de ocorrer uma partição. As bases de dados relacionais são um exemplo desta combinação, uma vez que elas não são tolerantes à partição, concluindo assim que existem algumas bases de dados relacionais com características semelhantes às bases de dados *NoSQL* (Holzschuher & Peinl, 2013). Esta abordagem não é adequada a grandes sistemas de bases de dados, pois, como não tolera partições não pode armazenar grandes quantidades de dados. Segundo (Fowler, 2012), os SGBDR tais como o *MySQL* ou *Postgres*, assim como as bases de dados orientadas a Colunas, *Vertica* são exemplos de produtos com estas características;

- **CP:** Este grupo de sistemas, apesar de ser consistente e tolerante à partição, não é uma boa opção, uma vez que o serviço nem sempre está disponível. Esta não disponibilidade surge da necessidade de, ao estar perante uma partição, os serviços afetados terem de esperar até que os dados fiquem consistentes, estando nesses momentos indisponíveis. É uma boa abordagem quando queremos um sistema consistente. Segundo (Fowler, 2012), bases de dados orientadas a colunas como o *Big Table*, *Hypertable*, *HBase*, bases de dados orientados a documentos como o *TerraStore* e ainda a base de dados Chave-valor *Berkeley DB* são exemplos de uma gama de produtos com estas características.

- **AP:** Este grupo de sistemas está sempre disponível, mesmo perante partições. Contudo um ou outro nodo pode ficar inconsistente após alguma operação. Este conjunto de propriedades pode também ser designado por "Eventual consistência" que iremos abordar num ponto mais à frente. Estes tipos de sistemas necessitam de uma estratégia para a resolução de conflitos de dados. Segundo (Fowler, 2012), bases de dados Chave-Valor como o *Dynamo* ou *Voldemort*, bases de dados orientadas a Colunas como o *Cassandra*, ou bases de dados orientadas a Documentos como o *SimpleDB* ou *CouchDB* são exemplos de uma gama de produtos com estas características.

Como podemos perceber, os três tipos de sistemas têm as suas vantagens e desvantagens, e compete a cada um escolher o tipo que mais se adequa à situação. Devemos assim, reconhecer quais dos requisitos do teorema *CAP* são mais importantes para o nosso sistema, para que o nosso sistema seja distribuído, escalável e com alta disponibilidade (Sousa, 2010).

3.2.2 *BASE* vs *ACID*

ACID e *BASE* representam duas filosofias diferentes de *design* em extremos opostos do espectro consistência e disponibilidade (Brewer, 2012). As partições de rede são raras, mas ocorrem espontaneamente em sistemas distribuídos. Como só podemos ter duas das três características que foram referidas anteriormente, torna-se necessário ter de escolher de entre a disponibilidade ou a consistência. Com o desenvolvimento da *web* 2.0, houve uma tendência para optar pela disponibilidade quando for possível tolerar alguma inconsistência temporária. Para descrever estas situações, surgiu o *BASE*. (Diana & Gerosa, 2010). A grande diferença entre *BASE* e *ACID* "está na consistência, enquanto as propriedades *BASE* focam-se no desempenho e na disponibilidade, abdicando um pouco da consistência, as propriedades *ACID* focam-se numa consistência forte em todas as operações" (Pritchett, 2008).

ACID

A fim de garantir a integridade dos dados, a maior parte dos sistemas de bases de dados convencionais são baseados em transações. Estas garantem a consistência dos dados em todas as situações de gestão de dados. Estas características são também conhecidas como *ACID* (Moniruzzaman & Hossain, 2013).

As transações *ACID* são, ao invés das transações *BASE*, pessimistas. Estão mais preocupadas com a segurança dos dados, ou seja, força a consistência em cada uma das suas operações. As bases de dados relacionais funcionam através de transações *ACID*. O acrónimo *ACID* significa:

- ***Atomicity*:** Ou todas as operações são bem sucedidas, ou nenhuma é;
- ***Consistency*:** A base de dados estará num estado consistente e sólido no início e no fim de uma transação;
- ***Isolation*:** As transações comportam-se como se fossem a única operação sendo executada na base de dados;
- ***Durability*:** Uma vez executada uma transação, esta é irreversível.

O conjunto destas transações garantem a confiabilidade do sistema (Vicknair et al., 2010), contudo surgem conflitos entre os diferentes aspetos de alta disponibilidade em sistemas distribuídos, uma vez que estes não são totalmente solucionáveis (Santos & Silva, 2014).

BASE

Os sistemas *BASE* surgiram com o aparecimento das bases de dados *NoSQL*, que estão a ignorar as exigências do teorema *CAP* a respeito da consistência, em prol de alcançarem melhores resultados nos requisitos de particionamento e disponibilidade (Santos & Silva, 2014). Quando se está perante um grande volume de dados distribuídos por várias bases de dados, as propriedades *ACID* são difíceis de alcançar, focando-se assim nas propriedades *BASE*. Neste padrão, a disponibilidade é alcançada através do tratamento de falhas locais, ou seja, uma falha local não pode ser aplicada a toda a rede. A disponibilidade do *BASE* é conseguida através do suporte de falhas parciais sem falhas totais do sistema, ou seja, se há uma falha numa base de dados que está particionada, apenas esta é afetada, continuando as restantes em pleno funcionamento (Pritchett, 2008). Funciona muito bem perante partições.

Como referido no ponto 3.2.1, ao contrário do padrão *BASE*, as transações *ACID* são pessimistas e forçam a consistência no fim de cada operação. Já o padrão *BASE* é otimista, pois aceita que os dados fiquem inconsistentes em determinados momentos.

BASE, segundo (Roe, 2012) significa:

- ***Basic Availability*:** O sistema garante a disponibilidade como está estipulado pelo Teorema *CAP*. O sistema dará uma resposta a qualquer pedido, mesmo que essa resposta seja uma "falha".

- ***Soft-State***: O sistema abandona completamente os requisitos de consistência do modelo *ACID*, ou seja, o estado do sistema pode mudar ao longo do tempo, mesmo durante os tempos em que não há transições ou *inputs*.
- ***Eventual Consistency***: Eventual consistência consiste no facto de em algum ponto no futuro os dados convergirão para um estado consistente. Ou seja, uma vez que o sistema recebe dados, estes irão ser propagados ao longo do tempo para os restantes pontos do sistema, podendo, no momento em que recebe os dados, não estar totalmente consistente. Contudo, mais tarde ou mais cedo, o sistema tornar-se-á consistente. Sempre que o sistema recebe uma entrada ou *input*, este não está verificando a sua consistência.

Os sistemas com propriedades *BASE* foram desenvolvidos como uma alternativa para produzir arquiteturas de dados mais escaláveis e acessíveis, fornecendo mais opções para expandir empresas/clientes de *TI* e simplesmente adquirir mais hardware para expandir operações de dados.

3.2.3 Tipos de Bases de Dados *NoSQL*

As bases de dados relacionais foram, durante muitas décadas, as escolhidas para armazenar grandes quantidades de dados devido ao seu alto desempenho e às suas propriedades *ACID*. Com o avanço do mundo *IT*, das redes sociais e dos dispositivos móveis cada vez mais interligados e conectados à rede, novos requisitos foram emergindo, tais como os serviços em *Cloud*. Com isto, vários tipos de bases de dados *NoSQL* surgiram e estão a ganhar cada vez mais popularidade (Tudorica & Bucur, 2011). Estes tipos de bases de dados são úteis quando se trabalha com uma enorme quantidade de dados e quando a sua natureza não requer um modelo relacional (Moniruzzaman & Hossain, 2013). Segundo o mesmo autor, "os sistemas *NoSQL* são distribuídos, são bases de dados não relacionais desenhadas para guardar dados em grande escala e para um processamento massivo de dados paralelos num grande número de servidores". Sendo assim, surgiram diversos tipos de bases de dados *NoSQL*. Estes novos tipos de bases de dados são conhecidos por não utilizarem a linguagem *SQL*, que era, até há pouco tempo, a linguagem quase universal nos SGBD.

Existem quatro tipos de bases de dados *NoSQL*, nomeadamente as bases de dados chave-valor, as bases de dados orientadas a documentos, as bases de dados orientadas a grafos, e as bases de dados orientadas a colunas (Tudorica & Bucur, 2011) (Tauro et al., 2012). (Moniruzzaman & Hossain, 2013)

defendem que as bases de dados *NoSQL* estão distribuídas por quatro categorias, sendo que cada uma é adequada a diferentes tipos de aplicações. Um outro conjunto de autores (Nayak & Poriya, 2013), defendem que existem 5 tipos de bases de dados *NoSQL* nomeadamente as bases de dados chave-valor, bases de dados orientadas a colunas, bases de dados orientadas a documentos, bases de dados em grafos e ainda as bases de dados orientadas a objetos. Esta última é baseada na programação orientada a objetos, e não é aceite pela maior parte da comunidade científica como uma base de dados *NoSQL.*

Apesar de todas estas diferentes perspetivas sobre as categorias e tipos de bases de dados *NoSQL*, apresentaremos as 4 categorias que são genericamente aceites pela comunidade científica – base de dados chave-valor, base de dados orientadas a documentos, base de dados orientadas a colunas e base de dados em grafos.

Bases de Dados Chave-Valor

As bases de dados chave-valor são as mais indicadas para tratar grandes volumes de dados com pouca complexidade (Buerli & Obispo, 2012). Este tipo de base de dados apesar de ser um modelo bastante simples, é um modelo poderoso. De uma maneira geral, este tipo de base de dados é composto por um conjunto de chaves que estão associadas a um único valor, que tanto pode ser uma *string* como um número binário.

As suas principais características são o *schema-free*, ou seja, não há um esquema fixo, sendo que podemos adicionar dados à base de dados sem entrar em conflito com outros dados já existentes, bastando para o efeito utilizar um grupo de pares chave-valor. Todos os dados são independentes uns dos outros. A principal vantagem deste modelo é a sua velocidade de consulta, a sua capacidade de armazenamento e a sua alta concorrência.

A simplicidade deste tipo de bases de dados torna-as ideais para consultas que se querem extremamente rápidas e altamente escaláveis, sendo assim aconselhadas para aplicações de gestão de perfis ou de utilizadores, assim como para recuperar nomes de produtos (Moniruzzaman & Hossain, 2013). Segundo (Atikoglu & State, 2012), "este modelo de base de dados é um componente vital em muitas empresas, principalmente em redes sociais, em lojas online e em análise de risco".

A figura 5 contém um exemplo de aplicação deste modelo.

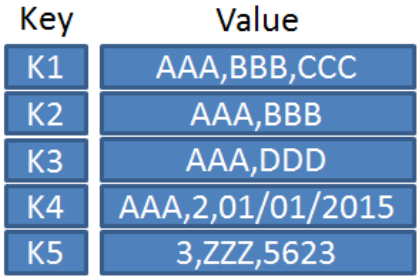

Figura 5 - Base de dados Chave-Valor.

Como se pode verificar, este modelo é um conjunto de tabelas, identificadas por uma chave que contém um conjunto de valores. Embora parecido ao modelo relacional, este modelo não contém qualquer ligação entre as tabelas nem contém chaves estrangeiras ou "*joins*".

Segundo o portal (*DBEngine*, 2017), em janeiro de 2017, os produtos mais representativos das bases de dados chave-valor existentes no mercado podem ser visualizados na tabela 1.

Tabela 1 - Base de dados Chave-Valor mais representativas no mercado. Adaptado de (DBEngine, 2017).

***Rank* (Jan 2017)**	***Rank* (Jan 2016)**	**SGBD**	**Score (Jan 2017)**
1	**1**	*Redis*	118,70
2	**2**	*Memcached*	28,44
3	**3**	*Riak KV*	10,78

Bases de Dados Orientadas a Documentos

Este modelo de base de dados armazena e organiza dados como coleções de documentos. Esses documentos não são tabelas estruturadas e não têm um tamanho uniforme para cada registo (Leavitt, 2010). Este modelo oferece uma escalabilidade horizontal, isto é, podemos dividir os dados por diferentes servidores, oferece um excelente desempenho, e é livre de esquema, ou seja, não há confusão entre dados quando são adicionados novos dados (Nayak & Poriya, 2013). Apesar destas boas características do modelo, segundo (Tauro et al., 2012), "as bases de dados orientadas a documentos não estão preocupadas com o alto desempenho de leitura e escrita simultânea, mas sim em garantir o armazenamento de dados de grandes escalas e um bom desempenho de consulta". Neste modelo cada

documento tem uma chave identificativa, sendo esta uma simples *string,* ou uma *string* referenciada a uma *URL* ou caminho.

Este modelo de base de dados é frequentemente comparado às bases de dados chave-valor (Han & Haihong, 2011), contudo a diferença consiste no modo como os dados são armazenados. Este modelo são basicamente documentos com versões que são coleções de outras coleções de chaves-valor.

Este modelo difere bastante do modelo relacional, uma vez que no modelo relacional um produto pode ter dados espalhados por várias tabelas, enquanto que, neste modelo, temos toda a informação de um produto num único documento (Han & Haihong, 2011). Esta informação pode ser observada na figura 6.

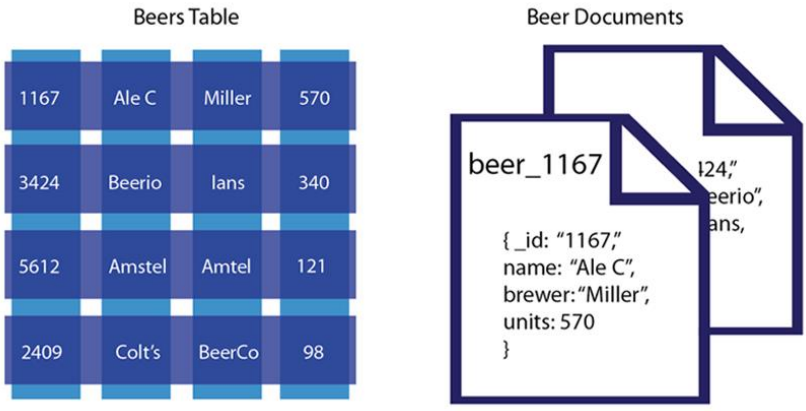

Figura 6 - Diferença entre as bases de dados relacionais e base de dados orientadas a documentos. Retirado de (Couchbase, 2016).

Neste modelo de base de dados, podem existir relações entre documentos, existindo, portanto, referências num documento a outros documentos.

O objetivo destas bases de dados é gerir e armazenar documentos, sendo que estes podem assumir vários formatos como o *XML*, *JSON* (*Javascript Object Notation*) ou *BSON* (*Binary JSON*) (Sousa, 2015). Segundo o portal *DBEngine* em janeiro de 2017, os produtos mais representativos das bases de dados orientadas a documentos existentes no mercado podem ser visualizados na tabela 2.

Tabela 2 - Bases de dados orientadas a documentos mais representativas no mercado. Adaptado de (DBEngine, 2017).

***Rank* (Jan 2017)**	***Rank* (Jan 2016)**	***SGBD***	***Score* (Jan 2017)**
1	**1**	*MongoDB*	331.90
2	**2**	*Amazon DynamoDB*	31.03
3	**3**	*Couchbase*	30.22

Como podemos verificar, o *MongoDB* destaca-se claramente da concorrência e é o produto mais representativo de entre as bases de dados orientadas a documentos. Estes diferentes produtos variam de uns para os outros no que diz respeito à consistência dos dados, disponibilidade, entre outros.

Bases de Dados Orientadas a Colunas

As bases de dados orientadas a colunas, assim como as bases de dados orientadas a documentos contêm uma estrutura de dados distribuída orientada a colunas que acomoda múltiplos atributos por uma chave (Moniruzzaman & Hossain, 2013). Estas diferem das base de dados relacionais, pois fazem praticamente uma inversão na organização dos dados (Leavitt, 2010). Os atributos são organizados em colunas, passando as linhas a conter as ocorrências de determinado atributo para cada instância de dados. Assim, as linhas armazenam um conjunto de atributos do mesmo tipo, ao passo que o conjunto de atributos de uma coluna contêm a informação de uma instância por completo. Na figura 7 podemos verificar essa situação.

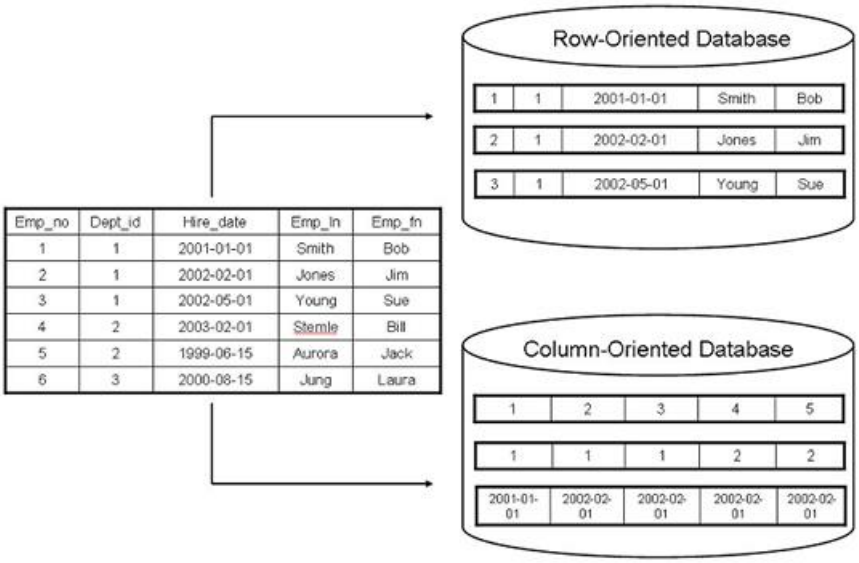

Figura 7 - Transformação para uma base de dados orientada a colunas. Retirado de (DBBest Technologies, 2017).

Esta abordagem de base de dados é projetada para tratar dados de forma não normalizada. Este modelo de base de dados foi desenvolvido de forma a suportar grandes quantidades de dados.

Este modelo é o indicado para situações em que estamos perante um enorme volume de dados e o número de operações *write* é superior às *read* (Abramova, Bernardino, & Furtado, 2014). Este modelo de base de dados não privilegia a consistência dos dados. Vários produtos foram desenvolvidos segundo este modelo de base de dados. Segundo o portal *DBEngine*, em janeiro de 2017, os produtos mais representativos das bases de dados orientadas a colunas existentes no mercado podem ser visualizados na tabela 3.

Tabela 3 - Bases de dados orientadas a colunas mais representativas no mercado. Adaptado de (DBEngine, 2017).

***Rank* (Jan 2017)**	***Rank* (Jan 2016)**	***SGBD***	***Score* (Jan 2017)**
1	**1**	*Cassandra*	136,44
2	**2**	*HBase*	59,14
3	**3**	*Accumulo*	3,47

O produto mais representativo de entre as bases de dados orientadas a colunas é o *Cassandra*. Este produto foi desenvolvido pelo *Facebook*, é *open-source* e é muito semelhante ao produto desenvolvido pela *Google*, o *BigTable* (Vicknair et al., 2010), sendo até inspirado neste. O Cassandra destaca-se pelo esquema ser muito flexível e por ter alta disponibilidade (Han & Haihong, 2011).

As bases de dados orientadas a grafos foram inspiradas no problema das sete pontes de *Konigsberg*, que foram primeiramente resolvidas pelo matemático suíço *Leonhard Euler* em 1736 (Buerli & Obispo, 2012). Este problema consistia em percorrer todas as pontes passando uma única vez em cada uma delas (Santos & Silva, 2014). Na figura 8 podemos visualizar o problema das pontes, assim como o caminho ou grafo construído por Euler.

Figura 8 - Problemas das pontes de Konigsberg e a sua solução. Retirado de (Santos & Silva, 2014).

A investigação em bases de dados orientadas a grafos foi bastante popular no início da década de 1990, mas caiu em desuso por causa do *Hypertext* e da investigação em *XML* (Vicknair et al., 2010). Contudo, devido ao avanço da *web* 2.0, das redes sociais e dos dados interligados e interconectados, surgiu um interesse em desenvolver e estudar o armazenamento de dados em grafos, levando a que este tipo de base de dados fosse ganhando cada vez mais popularidade (Holzschuher & Peinl, 2013). A *web*, na sua totalidade, é essencialmente um grafo de dados e informações ligadas em conjunto (Cudré-Mauroux & Elnikety, 2011).

Recentemente, esta área tem ganho a atenção de muitos especialistas por causa de muitos projetos que estão na moda e onde é necessário armazenar, interligar e retirar valor da informação (Buerli & Obispo, 2012). Isto só é possível se a sua representação for feita em forma de grafos. Sendo assim, a modelação em grafos é uma alternativa viável aos modelos relacionais e são ideais para representar dados da Química, Biologia, Semântica Web e Redes Sociais (Miller, 2013). São muito úteis na química e biologia, pois o seu modelo permite que se faça descobertas e análises de fármacos, uma vez que as suas operações são baseadas no reconhecimento de padrões (Buerli & Obispo, 2012). Nas redes sociais, os grafos são muito populares no que toca à investigação. Os grafos, não só guardam pessoas em nodos,

como também fotografias, mensagens, publicações, como ainda permitem relacionar todos estes nodos (Buerli & Obispo, 2012). Permitem encontrar o caminho mais perto entre dois nodos, sugerir amigos baseando-se nas características de um utilizador das redes sociais, e permitem também detetar padrões entre nodos ou relações. Estas consultas tornam-se muito mais simples quando realizadas numa base de dados orientada a grafos do que num modelo relacional. Na figura 9, podemos visualizar um modelo que nos mostra como podemos representar uma pequena rede social numa base de dados em grafos:

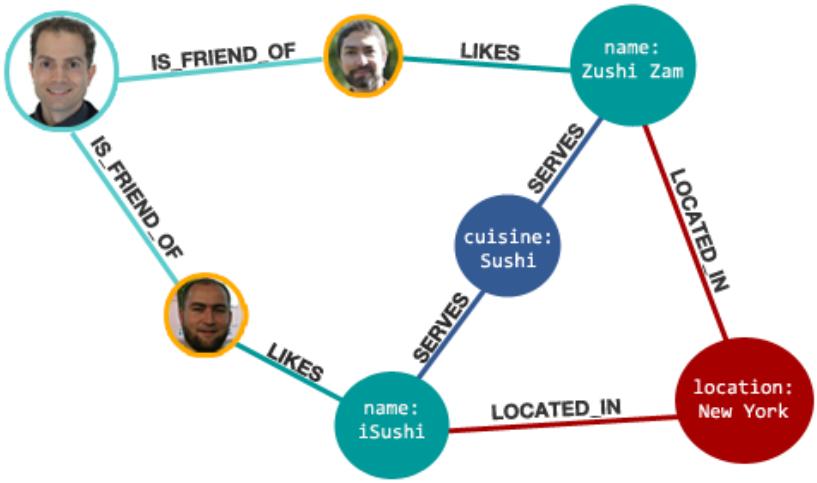

Figura 9 - Representação de uma rede social em grafo. Retirado de (Neo4j, 2013).

Como podemos verificar no modelo, a visualização e interpretação deste modelo é fácil e intuitiva. Nele estão representados vários nodos com diferentes tipos (pessoas, nomes, tipo de cozinha, localizações), e diversas relações entre estes.

O modelo de base de dados orientados a grafos é baseado na teoria de grafos e é composto por nodos (vértices) e relações (arestas) (Neubauer & Rodriguez, 2010).

- **Nodos**: Representam entidades que podem ser pessoas, contas, negócios, objetos, entre muitos outros. Os nodos podem ter propriedades e *labels*(tipos de nodos) diferentes. As propriedades servem para atribuir características a cada nodo, enquanto que as *labels* servem para distinguir diferentes tipos de nodos. Os nodos são equivalentes ao registo de uma linha numa base de dados relacional, ou de um documento numa base de dados orientada a documentos. Na Figura 9, estão representados sete nodos, com quatro *labels* diferentes. Uma das *labels* são pessoas e estão representados em três nodos. Uma outra *label, por exemplo,* é uma localização, neste caso *New York*.

- **Relações**: As relações conectam os nodos, e representam relações entre eles. Esta abstração de dados não é diretamente representada noutros modelos de bases de dados, sendo uma das características que diferencia este modelo de base de dados de outros modelos. Com as relações podemos determinar padrões que não são facilmente visíveis noutros modelos de representação de base de dados. Além de especificar o tipo de relação, podem também especificar propriedades dessa relação. Na Figura 10, estão representadas 6 relações entre os 3 nodos. Como existem relações iguais, pode-se verificar que apenas existem 3 tipos de relações (*Knows*, *is_member*, e *Members*). Estas relações têm propriedades como o "*Id*" ou o "*Since*", que nos devolvem mais informação sobre essas relações.

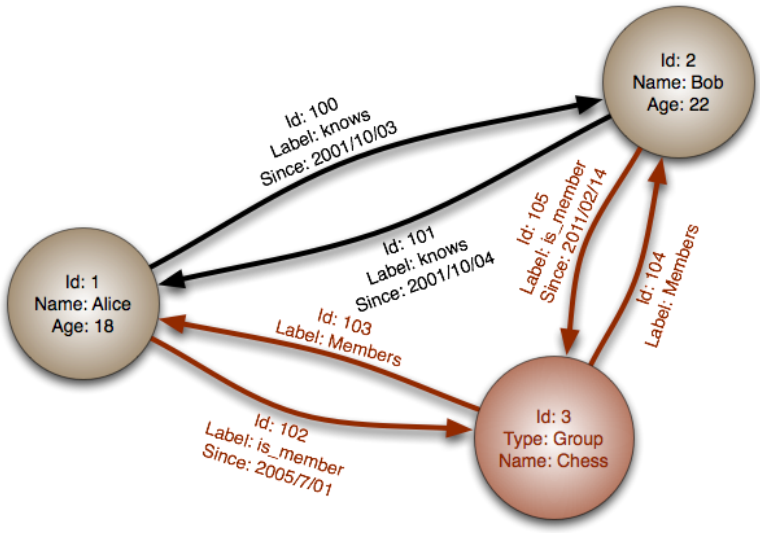

Figura 10 - Nodos e propriedades de um grafo. Retirado de (JDStraughan, 2013).

Segundo (Robinson, Webber, & Eifrem, 2015), o modelo de dados em grafo tem as seguintes características:

- Contém nodos e relações. Os nodos contêm propriedades, sendo que este conjunto pode ser comparado às bases de dados chave-valor;
- Cada nodo só pode pertencer a uma e uma só *label* (camada);
- As relações têm um nome e são diretas, tendo sempre um início e um fim;
- Pode haver relações com o próprio nodo, ou seja, com início e fim do mesmo nodo;
- Pode haver relações unidirecionais e bidirecionais;
- As relações também podem conter propriedades.

Esta é uma ferramenta poderosa para modelação de dados quando existe um foco nas relações entre entidades. Este desenho permite a construção de modelos preditivos e ajuda a descobrir correlações e padrões (Silvescu, Caragea, & Atramentov, 2010). Devido ao seu modelo de dados ser altamente dinâmico, através do qual todos os nodos estão ligados por relações, este permite percursos rápidos entre nodos, permitindo assim consultas bastante rápidas, quando comparado com o modelo relacional, uma vez que não necessita de muitos "*Joins*" (Rodriguez & Neubauer, 2011).

Os grafos são uma das mais úteis estruturas para modelação de objetos e interações (Vicknair et al., 2010). No modelo relacional é muito complicado representar relações, e como nos dias de hoje a informação se encontra toda relacionada, o desenvolvimento e adoção das bases de dados em grafos tem sido cada vez maior. Quando queremos fazer consultas numa base de dados relacional que envolve muitos relacionamentos, normalmente essa *querie* contém muitos *Joins*, ficando esta muito pesada para o sistema, fazendo com que o tempo de consulta à base de dados seja muito alto.

Segundo o portal *DBEngine*, em janeiro de 2017, os produtos mais representativos das bases de dados em grafos existentes no mercado podem ser visualizados na tabela 4.

Tabela 4 - Bases de dados em grafos mais representativas no mercado. Adaptado de (DBEngine, 2017).

***Rank* (Jan 2017)**	***Rank* (Jan 2016)**	***SGBD***	***Score* (Jan 2017)**
1	**1**	*Neo4j*	36,26
2	**2**	*OrientDB*	5,81
3	**3**	*Titan*	2,37

Como se pode constatar, o produto *Neo4j* é o mais representativo de entre as bases de dados em grafos. O produto *OrientDB*, é considerado multimodelo, ou seja, apesar de ter um modelo de base de dados em Grafo, é também considerado parte de base de dados orientada a documentos e chave-valor. O produto *Titan*, assim como o *Neo4j*, tem apenas e só uma modelação em grafo.

Num outro artigo (Shimpi, 2013), é realizado um apanhado de alguns produtos representativos das bases de dados em grafo, onde são comparados tendo em conta a sua linguagem de consulta, o seu uso, o tipo de grafo, as suas aplicações entre outras. A comparação encontra-se na tabela 5.

Tabela 5 - Comparação de bases de dados em grafo. Adaptado de (Shimpi, 2013).

Base de Dados em Grafo	Protocolo	API	Modelo de Disponibilidade	Linguagem de Consulta	Acessibilidade	Tipo de Grafo	Portabilidade	Aplicação
Allegrograph	*REST*	*Java*	Não *Open source*	*SPARQL*	Tamanho Fixo	Simples	Não	Raciocínio geotemporal e análise de redes sociais.
DEX	-	*C++; Java*	Não *Open source*	Baseado em *SQL*	Tamanho Fixo; Caminho mais curto	Atribuído	Sim	Gestão de grandes quantidades de Grafos.
Infinitegraph	*REST*	*Java*	Não *Open source*	*Gremlin*	Tamanho fixo, regular e simples. Caminho mais curto	Atribuído	Não	Inteligência governamental, social e empresarial com análise de Grafos.
Hypergraph	*REST/ JSON*	*Java*	*Open source*	Semelhante a *SQL*	-	Simples	Sim	Representação de Conhecimento, Inteligência Artificial e Bioinformática.
Neo4j	*REST/ JSON*	*Java; Jpython; Jruby*	*Open source*	*Cypher*	Tamanho fixo, regular e simples. Caminho mais curto	Atribuído	Sim	Representação de todo o tipo de relacionamento

Base de Dados em Grafo	Protocolo	API	Modelo de Disponibilidade	Linguagem de Consulta	Acessibilidade	Tipo de Grafo	Portabilidade	Aplicação
								entre dados. *Web* semântica, Biologia.
Sones	*REST/ JSON*	*C#*	Não *Open source*	*GraphQL* (Baseado em *SQL*)	-	Simples	Sim	Manipulação de Dados semiestruturados.
VertexDB	*HTTP/ JSON*	*C; C++*	Não *Open source*	*SQL* através de *JDB*	Tamanho fixo, regular e caminho simples	Simples	Sim	Armazenamento de Grafos.
G-Store	*REST*	*C/C++*	Não *Open source*	*SQL*	Tamanho fixo, regular e simples. Caminho mais curto.	Simples, atribuído	Sim	Livraria de armazenamento de Dados.
OrientDB	*REST/ JSON*	*Java*	*Open source*	*SQL*	Tamanho fixo, regular e simples. Caminho mais curto.	Simples, atribuído.	Sim	Para Bases de Dados orientadas a Documentos
Infogrid	*REST/ JSON*	*Java*	*Open source*	Interface *Web* em *HTML*	-	Simples	Sim	Para aplicações *WEB*.
Cloudgraph	-	*C#*	Não *Open source*	*GQL*	-	Simples	Sim	Baseado na *Web*. Para rápido acesso a conjuntos de

Base de Dados em Grafo	Protocolo	API	Modelo de Disponibilidade	Linguagem de Consulta	Acessibilidade	Tipo de Grafo	Portabilidade	Aplicação
								dados interconectados.
Trinity	*C# language binding*	*C#*	Não *Open source*	*SPARQL*	Tamanho Fixo, caminho mais curto	Simples	Sim	Armazenamento de Grafos e processamento online de *Queries*.

Neste mesmo artigo, *Shimpi* refere que o melhor produto e mais utilizado de entre as bases de dados em Grafo é o *Neo4j*.

Não existe até ao momento uma linguagem universal para este tipo de base de dados (Miller, 2013). As linguagens mais utilizadas são o *Cypher*, *Gremlin*, *SPARQL*, e *RDF*, onde muitas utilizam *APIs* onde podem aceder e manipular os dados utilizando linguagens como *JAVA, C+, e C#* (Shimpi, 2013).

Segundo (Buerli & Obispo, 2012), estes quatro tipos de bases de dados podem ser comparados entre si relativamente ao tamanho de dados e à sua complexidade. Segundo o mesmo autor, as bases de dados chave-valor são as que têm um melhor desempenho perante uma maior quantidade de dados. Para dados não tão volumosos, mas muito complexos, a base de dados mais indicada são as baseadas em grafos. Na figura 11, podemos ver o seu esquema:

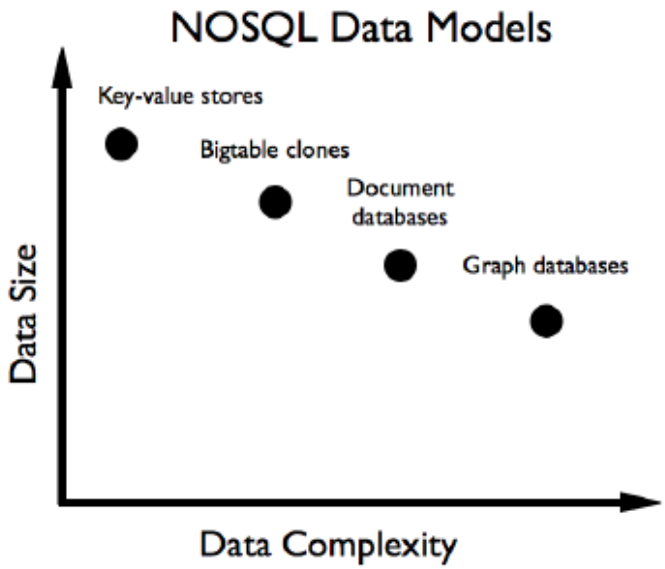

Figura 11 - Modelos de dados NoSQL. Retirado de (Buerli & Obispo, 2012).

3.2.4 Bases de Dados *NoSQL* versus Relacionais

Com o desenvolvimento da *Internet* e da computação em *Cloud*, houve uma necessidade de bases de dados que pudessem armazenar grandes volumes de dados de forma eficaz, uma procura por alto desempenho nas leituras e nas escritas, factos que, as bases de dados relacionais não estavam a conseguir acompanhar.

A procura por bases de dados *NoSQL* têm tido cada vez mais procura devido aos seguintes aspetos (Han & Haihong, 2011):

- Alta concorrência de leitura e escrita com baixa latência;
- Armazenamento eficiente de grandes volumes de dados;
- Alta escalabilidade e alta disponibilidade;
- Baixos custos operacionais e de gestão comparativamente com os SGBDR;
- Leituras e consultas lentas nas bases de dados relacionais;
- Capacidade limitada das bases de dados relacionais;
- Dificuldade de expansão das bases de dados relacionais;

A principal razão para o desenvolvimento de novas tecnologias relacionais de base de dados, como vimos anteriormente, são a sua capacidade limitada e a sua dificuldade de expansão. Contudo existem outras limitações da tecnologia relacional que proporcionaram o desenvolvimento de outro tipo de base de dados. Algumas dessas limitações são (Leavitt, 2010):

- **Escalabilidade**: As bases de dados relacionais têm muitas dificuldades em trabalharem quando se encontram distribuídas por vários computadores. As bases de dados relacionais não são projetadas para funcionar com o particionamento dos dados;
- **Complexidade**: Nas bases de dados relacionais é muito difícil gerir a complexidade, isto porque todos os dados devem ser convertidos em tabelas. Quando algum dado não se encaixa em nenhuma tabela, a estrutura da base de dados pode ser complexa, difícil e lenta para trabalhar.
- ***SQL***: A linguagem *SQL* é direcionada para trabalhar com dados estruturados, que são facilmente encontrados nas bases de dados relacionais. Contudo, em dados não estruturados, a linguagem *SQL* não funciona bem uma vez que pode envolver grandes quantidades de código complexas.
- **Grandes conjuntos de recursos**: As bases de dados relacionais oferecem uma grande variedade de recursos quanto à integridade dos dados. No entanto, as bases de dados *NoSQL* não precisam de todos os recursos, bem como do custo e da complexidade que eles adicionam.

Apesar de as bases de dados *NoSQL* parecerem desempenhar melhor que as bases de dados relacionais, também têm algumas vantagens e desvantagens. (Leavitt, 2010) e (Nayak & Poriya, 2013) citam algumas delas:

Vantagens das bases de dados *NoSQL*:

- As bases de dados *NoSQL* geralmente processam dados mais rapidamente do que as bases de dados relacionais;
- São mais facilmente escaláveis que as bases de dados relacionais;
- Alguns produtos *NoSQL* como o *Riak* ou *Cassandra* são programados para lidar com falhas de *hardware*;
- São mais rápidas, eficientes e flexíveis que as bases de dados relacionais;
- Têm vindo a evoluir cada vez mais.

Desvantagens das bases de dados *NoSQL*:

- As bases de dados *NoSQL* são ainda um pouco imaturas;
- Não existência de uma linguagem padrão, ao contrário das bases de dados relacionais que têm o *SQL* como linguagem base;
- Existência de algumas bases de dados *NoSQL* não compatíveis com as transações *ACID*;
- A manutenção deste tipo de bases de dados é difícil de realizar.

3.3 Bases de Dados em Grafos

Como pudemos constatar, os três produtos mais representativos das bases de dados baseadas em grafos são o *Neo4j*, o *OrientDB* e o *Titan*. Neste ponto abordamos sucintamente estes três produtos, de modo a decidir e a justificar a escolha de um deles que é utilizado neste projeto de dissertação.

3.3.1 Neo4J

O *Neo4j* (figura 12) é um produto desenvolvido pela *Neo Technology*, para armazenamento de dados numa estrutura baseada na teoria dos grafos. É atualmente o produto mais utilizado de entre as bases de dados em grafo tendo clientes como o *Ebay*, *LinkedIn*, *Infojobs* e *Walmart*. *Neo4j* é uma base de dados escalável, construída não só para armazenar dados, mas especialmente os seus relacionamentos. O mecanismo de processamento e armazenamento do *Neo4j* oferece um bom desempenho em tempo real, ajudando as organizações a criar aplicações inteligentes para atender aos atuais desafios de dados em constante evolução.

Figura 12 - Neo4j.

Modelo de Dados

O modelo de dados do *Neo4j*, assim como quase todos os restantes produtos representativos das bases de dados em grafos é constituído por nodos, relações e propriedades (Vukotic & Watt, 2013). Os dados são representados nos nodos, nas relações e nas propriedades, sendo que os nodos e as relações contêm propriedades.

As relações podem ser bidirecionais ou unidirecionais, sendo que existe sempre um nodo de partida e um nodo de chegada. Contudo, o nodo de partida e chegada podem ser o mesmo. Não é possível criar relações sem uma direção.

O *Neo4j* usa o armazenamento nativo em grafo com o *Native GPE*, que é um mecanismo de processamento de Dados.

Modelo de Consulta

A base de dados *Neo4J* suporta acessos *CRUD* (*Create*, *Read*, *Update* e *Delete*) em várias linguagens. Não existe uma linguagem padrão, contudo existem várias implementações e estruturas diferentes que variam entre a *API Java* do *Neo4j*, a interface *REST*, uma linguagem *DSL* chamada *Gremlin* e outra chamada *Cypher* (Miller, 2013). Existem vários estudos comparativos entre estes diferentes meios de consulta. Segundo (Holzschuher & Peinl, 2016), apesar de muitos indicadores apontarem a linguagem *Cypher* como a melhor linguagem de consulta, esta não é totalmente confirmada no seu estudo. Contudo, a linguagem *Cypher* foi a que obteve melhores resultados comparativamente ao acesso nativo para *API* do *Java* e da linguagem *Gremlin*. O pior resultado foi alcançado pela *API Restful* do *Java*. Vários autores referem que a linguagem *Cypher* está a tornar-se a linguagem padrão do *Neo4j* (Angles, 2016). O *Cypher* tem diversas vantagens comparativamente aos restantes concorrentes, pois é de fácil compreensão, fácil de aprender, é uma linguagem declarativa e é bastante semelhante à linguagem *SQL*. Nenhuma das outras linguagens é tão clara em sintaxe e semântica como o *Cypher* (Holzschuher & Peinl, 2016).

Muitas bases de dados populares oferecem soluções chamadas de *Sharding*, isto é, dividindo dados em vários servidores. Contudo, existe um problema matemático em dividir um conjunto de grafos de forma ótima num conjunto de servidores, o que torna esta divisão quase impossível (Montag, 2013).

Há duas partes em cada instância do *Neo4j*. Uma parte é a própria base de dados. O outro é o componente da gestão de *Clusters* (figura 13). O componente da gestão de Clusters mantém-se sempre sincronizado com todas as instâncias do cluster. Ele mantém o controlo sobre todas as instâncias que querem entrar ou sair do sistema. Quando a eleição do "pai" se torna necessária, o componente de gestão de *Clusters* garante automaticamente que o novo "pai" é consistentemente eleito. A camada da base de dados gere o resto do sistema. As instâncias do "filho" carregam atualizações transacionais do "pai".

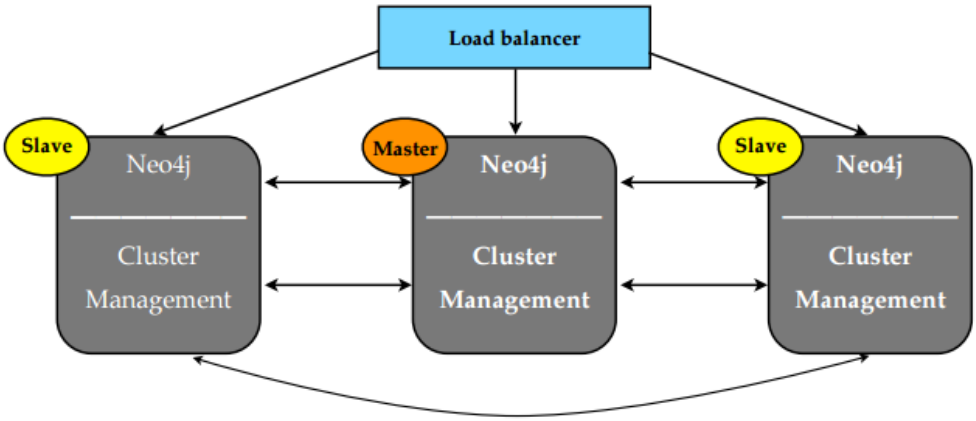

Figura 13 - Partição no Neo4j. Retirado de (Montag, 2013).

Com este tipo de partição conseguimos ter:

- Redundância total dos dados;
- Tolerância a falhas do serviço;
- Grafos linearmente escaláveis;

Este tipo de partição só está disponível na versão empresarial do *Neo4j*. O *Neo4j* com as suas características permite:

- **Proteger os dados:** Num *Cluster* de alta disponibilidade do *Neo4j*, o grafo é replicado para cada instância do cluster. Isso significa que o conjunto de dados completo é replicado em todo o cluster,

para cada servidor. Independentemente do número de instâncias que falham, todos os dados são mantidos seguros, enquanto uma instância permanece disponível.

- **Leitura escalável:** Embora as operações de escrita sejam executadas ao mesmo tempo no *cluster* "pai", as operações de leitura podem ser feitas localmente em cada "filho". Isso significa que a capacidade de leitura do *cluster* de alta disponibilidade aumenta linearmente com o número de servidores.

A *Neo Tecnology* fornece ainda algumas comparações do *Neo4j* com outras bases de dados *NoSQL* (tabela 7) e ainda com as bases de dados relacionais (tabela 6).

Tabela 6 - Bases de dados relacionais vs base de dados Neo4j.

	Base de Dados Relacional	Base de Dados *Neo4j*
Armazenamento de Dados	Armazenamento em tabelas fixas e pré-definidas, com linhas e colunas com dados conectados, muitas vezes desarticulados entre tabelas, prejudicando a eficiência da mesma.	Estrutura de armazenamento de grafo livre de índice que resulta em transações e processamento mais rápido para relacionamento de dados.
Modelação de Dados	O modelo de base de dados deve ser desenvolvido com modeladores e traduzido de um modelo lógico para um modelo físico. Uma vez que o tipo de dados e fontes devem ser conhecidos antes do tempo, quaisquer alterações levam a perdas de tempo desnecessárias	Modelo de dados flexível sem discordância entre o modelo lógico e físico. Os tipos de dados e fontes podem ser adicionados ou alterados a qualquer momento, levando a tempos de desenvolvimentos relativamente curtos.
Desempenho de Consulta	O desempenho do processamento de dados sofre com o número e a profundidade de *joins*. Quantos mais *joins* mais lento se torna a consulta.	O processamento de grafos garante zero de latência e desempenho em tempo-real, independentemente do número ou profundidade de relacionamentos.
Linguagem de Consulta	*SQL*: Uma linguagem de consulta que aumenta em complexidade com o número de *joins* necessários para consultas de dados conectadas.	Dispõe de algumas, contudo o *Cypher* é a mais utilizada. É uma linguagem de consulta gráfica que fornece a maneira mais eficiente e expressiva para descrever consultas de relacionamento.

	Base de Dados Relacional	Base de Dados *Neo4j*
Suporte às Transações	Transações *ACID*	Transações *ACID* para dados totalmente consistentes e confiáveis. Ideal para aplicações empresariais globais sempre atuais.
Processamento em Escala	Escalável através da replicação. Escalar a arquitetura é possível, mas bastante dispendioso.	A escalabilidade é mantida por meio da replicação.
Eficiência do Centro de Dados	A consolidação do servidor é possível, mas custosa para a arquitetura escalonada. Reduzir a arquitetura é caro em termos de compra, uso de energia e tempo de gerenciamento.	Os dados e as relações são armazenados nativamente, melhorando o desempenho à medida que a complexidade e escala crescem. Isto leva à consolidação de servidores e uso mais eficiente de hardware.

A comparação entre a generalidade das outras bases de dados *NoSQL* e a base de dados *Neo4j* pode ser encontrada na tabela 7:

Tabela 7 - Outras bases de dados NoSQL vs base de dados Neo4j.

	Outras Bases de Dados *NoSQL*	Base de Dados *Neo4j*
Armazenamento de Dados	O desempenho e a confiabilidade de dados ficam piores com a escala e a complexidade de conexões.	O facto de os grafos serem livres de índice resulta em transações e processamento mais rápidos para relacionamentos de dados.
Modelação de Dados	Modelo de dados não adequado para arquiteturas de empresas como as bases de dados orientadas a colunas e as bases de dados orientadas a documentos que não oferecem controlo ao nível do *design*. Coloca uma pressão indevida no nível de aplicação para capturar e resolver problemas.	Modelo de dados flexível e intuitivo que facilita a comunicação entre desenvolvedores, arquitetos e administradores de base de dados.
Desempenho de Consulta	Não têm capacidade de processamento de grafos para relacionamentos de dados, portanto, todos os relacionamentos devem ser criados no nível aplicacional.	O processamento de grafos garante zero de latência e desempenho em tempo-real, independentemente do número ou profundidade de relacionamentos.

	Outras Bases de Dados *NoSQL*	Base de Dados *Neo4j*
Linguagem de Consulta	Várias linguagens de consulta, contudo não existe nenhum ideal para expressar relações de dados.	Dispõe de algumas, contudo o *Cypher* é a mais utilizada. É uma linguagem de consulta gráfica que fornece a maneira mais eficiente e expressiva para descrever consultas de relacionamento.
Suporte às Transações	Transações *BASE* levam à corrupção de dados porque a disponibilidade básica e a eventual consistência não são confiáveis para os relacionamentos de dados.	Transações *ACID* que garantem que os dados sejam totalmente consistentes e confiáveis 24h por dia – o que torna esta base de dados perfeita para aplicações corporativas globais sempre atuais.
Processamento em Escala	A escalabilidade depende da arquitetura de escala que não protege a integridade de dados semelhantes a grafos, portanto os dados não são confiáveis.	A arquitetura mantém a escalabilidade por meio da replicação.
Eficiência do Centro de Dados	Expandir a arquitetura implica ter mais hardware, o que irá aumentar os custos de energia, vulnerabilidades da rede e outros riscos.	Os dados e as relações são armazenados nativamente, melhorando o desempenho à medida que a complexidade e escala crescem. Isto leva à consolidação de servidores e o uso mais eficiente de hardware.

3.3.2 Titan

Titan (figura 14) é outros dos produtos representativos de bases de dados em grafos. Foi desenvolvido pela empresa *Aurelius*. Este produto de base de dados dispõe de muitas características próprias que o distinguem de todos os outros produtos de bases de dados em grafos e está otimizado para *clusters* distribuídos. Foi lançado no ano de 2012, sendo que é uma ferramenta *open source* e foi implementado na linguagem *Java*, no entanto, suporta várias linguagens de programação tais como *Clojure*, *Java* e *Python*.

Este produto de base de dados usa *clusters* de multi-máquinas distribuídas o que faz com que o *Titan* possa ser escalado até tamanhos enormes de dados, podendo suportar grafos de mais de 100 biliões de nós e dezenas de utilizadores acedendo em simultâneo. Este tipo de base de dados é normalmente utilizado com outros *SGDB* tais como o *Apache Cassandra*, o *HBase* ou o *Oracle BerkeleyDB*.

Figura 14 - Titan

Modelo de Dados

O *Titan* está focado na modelação de dados em grafos e na execução eficiente de consultas. O *Titan* implementa interfaces robustas e modulares para a persistência dos dados, indexação de dados e acesso ao cliente.

Os dados no *Titan* são alojados num esquema constituído por nodos, propriedades dos nodos, relações e propriedades das relações.

Modelo de Consulta

O *Titan* aceita acessos *CRUD* (*Create*, *Read*, *Update* e *Delete*) e estes podem ser realizados através da linguagem *Gremlin* (Holzschuher & Peinl, 2013). Esta linguagem desempenha bastante bem em *queries* simples, sendo considerada uma linguagem de baixo nível (Holzschuher & Peinl, 2016). Em várias comparações de desempenho, esta tem desempenhado pior do que a principal linguagem de consulta utilizada pelo *Neo4J* (Holzschuher & Peinl, 2013) (Wood, 2012) (de Oliveira Santos, et al., 2016).

As consultas podem ser realizadas também através de uma *API* do Java.

Arquitetura do Sistema

O produto *Titan* suporta transações *ACID*. Uma vez que este produto é utilizado em conjunto com outros produtos de base de dados, nomeadamente o *Cassandra*, *Apache HBase* e *Oracle Berkeley DB*, o teorema *CAP* deve ser considerado. As suas compensações em relação ao teorema *CAP* são representadas na figura 15.

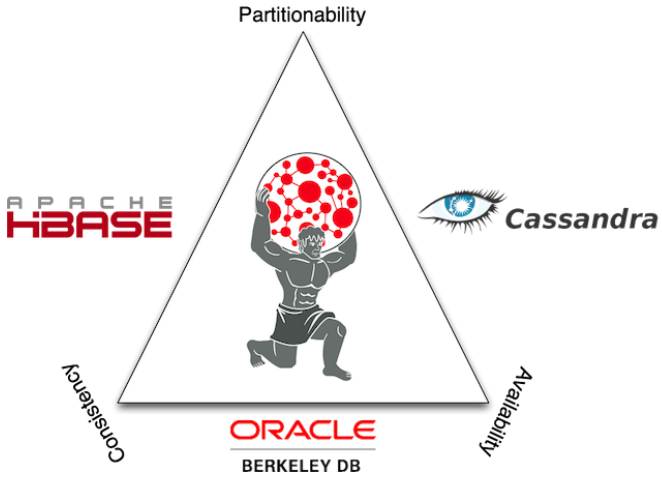

Figura 15 - Base de dados Titan e o teorema CAP. Retirado de (Tesoriero, 2013).

Como podemos constatar, o *Titan* quando utilizado com o *Berkeley DB* (que é um sistema não distribuído) privilegia a consistência dos dados e a disponibilidade. O *Titan* com o *HBase* privilegia a consistência e o particionamento, e com o *Cassandra* a disponibilidade e o particionamento.

O *Titan* foi concebido para escalar horizontalmente em vários servidores e em vários dispositivos, pelo que se torna necessário o uso do *Cassandra* e do *HBase* que também são sistemas de bases de dados escaláveis. Assim, é aproveitada toda a escalabilidade dos dois sistemas de modo a garantir a distribuição dos dados, a replicação, o *backup* e a tolerância a falhas.

A arquitetura do *Titan* permite trabalhar com uma ampla gama de tecnologias de armazenamento, índice e clientes.

3.3.3 OrientDB

OrientDB (figura 16) é um sistema de gestão de base de dados *NoSQL open-source* desenvolvido em *Java* e é considerado multimodelo, ou seja, é orientado tanto a documentos como a grafos (Shimpi, 2013). Esta base de dados tem o melhor de uma base de dados em grafo distribuída e a flexibilidade de uma base de dados orientada a documentos num só produto (Barmpis & Kolovos, 2014). O *OrientDB* afirma-se como "a primeira e mais escalável base de dados *NoSQL* de alto desempenho (OrientDB, 2017). O *OrientDB* suporta as mais conhecidas linguagens de programação, tais como *.Net*, *C*, *C#*, *C++*, *Java*, *JavaScript*, *PHP*, *Python*, etc. A nível de transações estas são *ACID* (Tesoriero, 2013).

O *OrientDB* tem várias vantagens tais como:

- **Multi-Modelo de Base de Dados:** Contém num único produto um mecanismo de uma base de dados distribuída em grafos com a flexibilidade de uma base de dados orientada a documentos;
- **Replicação *Multi-Master*:** A base de dados pode ser distribuída por vários servidores que utilizam o sistema de replicação. Os dados dos grafos podem ser adicionados sem configuração complexa. Funciona perfeitamente em *Cloud*.
- ***SQL*:** O *OrientDB* suporta a linguagem *SQL* através de extensões para manipular os grafos.
- **Transações Distribuídas:** As transações são *ACID* e distribuídas através de diferentes servidores.
- **Sincronização com os SGBDR:** Com a versão *OrientDB Teleporter* pode-se sincronizar (ou migrar) a base de dados relacional do *Oracle*, *SQL Server*, *PostgreSQL* para o *OrientDB*.
- **Utilização gratuita:** A utilização do *OrientDB Community Edition* é gratuita, mesmo para fins comerciais.

Figura 16 – OrientDB

Modelo de Dados

O *OrientDB* desenvolveu um modelo de base de dados em grafo com um conceito multimodelo, isto é, uma maneira mais flexível de representar domínios complexos com base no modelo de Grafos, enriquecido com "*Documents*" (que provêm das bases de dados orientadas a documentos), *objetos* e chaves-valor (bases de dados chave-valor), entre outros (Tesoriero, 2013). Todos esses modelos são geridos pelo *OrientDB*. O *OrientDB* também pode ser usado apenas como uma base de dados em grafo normal como o *Neo4j* (Tesoriero, 2013).

Modelo de Consulta

As duas linguagens mais comuns para consulta no *OrientDB* são o *Gremlin* e o *SQL* (Tesoriero, 2013). O *SQL* é a linguagem de consulta mais conhecida a nível mundial, enquanto que o *Gremlin* é uma linguagem especializada para base de dados em grafos (Tesoriero, 2013). *Gremlin* é baseado na linguagem de programação *Groovy* (Santos & Silva, 2014).

Arquitetura do Sistema

O *OrientDB* pode ser distribuído em diferentes servidores e usado de diferentes maneiras para alcançar o máximo de desempenho, escalabilidade e robustez. A arquitetura é *multi-master*, ou seja, cada servidor pode efetuar operações de leitura e escrita, onde depois existe a replicação. Isto é, se houver 3 servidores *Master* e 100 servidores "filho", cada operação de gravação será replicada em 103 servidores.

3.3.4 Escolha do Produto representativo

Por todas as razões apresentadas anteriormente e por diversos testes realizados por outros autores, o produto que estudamos mais profundamente e que é utilizado para fazer a comparação é o *Neo4J*. Este tem claramente vantagem em relação à concorrência. Tem uma linguagem de consulta de fácil aprendizagem, tem a maior quota de mercado das bases de dados em grafos que se traduz num maior suporte e material disponível e tem os melhores desempenhos dentro das bases de dados em grafo. Além disso, não está dependente do uso de outros sistemas de gestão de base de dados como o *Titan*. Dada a maior representatividade do produto *Neo4j* no âmbito das bases de dados em grafos (ver Tabela 4) é esta a selecionada para o projeto de dissertação, nomeadamente para os testes de comparação de desempenho com as bases de dados relacionais, que estão detalhados num ponto mais à frente.

4. Recolha, Análise, Tratamento e Carregamento dos Dados

Neste capítulo 4, é descrito como e onde foi realizada a recolha dos dados, as análises que foram realizadas para descobrir incongruências e defeitos nos dados, assim como o tratamento sobre estes realizado. No último ponto, será explicado como foi feito o carregamento dos cenários da base de dados relacional, assim como dos cenários da base de dados em grafos.

4.1 Recolha dos Dados

Os conjuntos de dados recolhidos fornecem informação sobre todos os voos comerciais com partida e chegada nos Estados Unidos da América realizados entre outubro de 1987 e abril de 2008. Estes dados foram fornecidos pela *American Statistical Association* baseado nos dados do *United States Department of Transportation*.

A escolha deste conjunto de dados deveu-se ao facto de ter sido utilizado por dois colegas em dissertações anteriores onde realizaram comparações com outro tipo de bases de dados *NoSQL*, e também pelo facto de os dados fornecidos poderem ser modelados de forma perfeita em grafos.

Foram fornecidos um total de 25 conjuntos de dados em formato *CSV* com um tamanho total de 11,2 *GB*, sendo que 22 desses conjuntos dizem respeito aos voos de um ano. Os conjuntos de dados encontram-se divididos por anos e podem ser visíveis na tabela 8.

Tabela 8 - Tamanho e número de registos dos datasets.

Nome do *Dataset*	Tamanho	Número de Registos
1987	121 MB	1 311 826
1988	477 MB	5 202 096
1989	463 MB	5 041 201
1990	485 MB	5 270 893
1991	468 MB	5 076 925
1992	469 MB	5 092 158
1993	468 MB	5 070 501
1994	478 MB	5 180 048
1995	506 MB	5 327 435
1996	509 MB	5 351 983
1997	515 MB	5 411 843

Nome do *Dataset*	Tamanho	Número de Registos
1998	513 MB	5 384 721
1999	527 MB	5 527 884
2000	543 MB	5 683 047
2001	572 MB	5 967 780
2002	505 MB	5 271 359
2003	597 MB	6 488 540
2004	638 MB	7 129 270
2005	639 MB	7 140 596
2006	640 MB	7 141 923
2007	670 MB	7 453 216
2008	657 MB	7 009 728
Airports	268 Kb	3 376
Carriers	45,6 Kb	1 491
Plane-data	423 Kb	5 029

Como se pode verificar através da tabela 8, existe um conjunto de dados com um total de 3 376 aeroportos distintos, 1 491 companhias aéreas, 5 029 aviões distintos, assim como 123 534 973 voos realizados.

4.2 Análise e Tratamento dos Dados

Uma vez recolhidos os dados, torna-se necessário efetuar uma análise, de modo a verificar a qualidade destes, e, caso necessário, efetuar algum tratamento aos mesmos. Esta análise é fundamental para perceber quais os dados imprescindíveis a utilizar nos testes de desempenho.

O conjunto de dados que nos fornece informações sobre os aviões, presente no *dataset Plane-data*, tem a estrutura presente na figura 17.

1	tailnum	type	manufacturer	issue_date	model	status	aircraft_type	engine_type	year
36	N10156	Corporation	EMBRAER	02/13/2004	EMB-145XR	Valid	Fixed Wing Multi-Engine	Turbo-Fan	2004
37	N102UW	Corporation	AIRBUS INDUSTRIE	05/26/1999	A320-214	Valid	Fixed Wing Multi-Engine	Turbo-Fan	1998
38	N10323	Corporation	BOEING	07/01/1997	737-3TO	Valid	Fixed Wing Multi-Engine	Turbo-Jet	1986
39	N103US	Corporation	AIRBUS INDUSTRIE	06/18/1999	A320-214	Valid	Fixed Wing Multi-Engine	Turbo-Fan	1999
40	N104UA	Corporation	BOEING	01/26/1998	747-422	Valid	Fixed Wing Multi-Engine	Turbo-Fan	1998
41	N104UW	Corporation	AIRBUS INDUSTRIE	07/02/1999	A320-214	Valid	Fixed Wing Multi-Engine	Turbo-Fan	1999
42	N10575	Corporation	EMBRAER	06/24/2003	EMB-145LR	Valid	Fixed Wing Multi-Engine	Turbo-Fan	2002

Figura 17 - Dataset Plane-Data.

A descrição dos atributos referentes aos aviões pode ser visualizada na tabela 9.

Tabela 9 - Atributos do dataset Plane-Data.

Atributo	Exemplo	Descrição do atributo
Talinum	N10156	Número do avião.
Type	Corporation	Tipo do avião.
Manufacturer	EMBRAER	Fabricante do avião.
Issue_date	02/13/2004	Data de emissão.
Model	EMB-145XR	Modelo do avião.
Status	Valid	Estado atual do avião.
Aircraft_type	Fixed Wing Multi-Engine	Tipo de avião.
Engine_type	Turbo-Fan	Tipo de motor.
Year	2004	Ano de construção.

O conjunto de dados que fornece informações sobre os aeroportos, presente no *dataset Airports*, tem a estrutura presente na figura 18.

	iata	airport	city	state	country	lat	long
2	00M	Thigpen	Bay Spring	MS	USA	31.953764	-89.23450472
3	00R	Livingston	Livingston	TX	USA	30.685861	-95.01792778
4	00V	Meadow L	Colorado S	CO	USA	38.945748	-104.5698933
5	01G	Perry-War	Perry	NY	USA	42.741346	-78.05208056
6	01J	Hilliard Ai	Hilliard	FL	USA	30.688012	-81.90594389
7	01M	Tishoming	Belmont	MS	USA	34.491666	-88.20111111

Figura 18 - Dataset Airports.

A descrição dos atributos referentes aos aeroportos pode ser visualizada na tabela 10.

Tabela 10 - Atributos do dataset Airports.

Atributo	Exemplo	Descrição do atributo
IATA	00M	Código IATA do aeroporto.
Airport	Thigpen	Nome do aeroporto.
City	Bay Springs	Cidade do aeroporto.
State	MS	Estado do aeroporto.
Country	USA	País do aeroporto.
Lat	31.95376472	Latitude do aeroporto.
Long	-89.23450472	Longitude do aeroporto.

O conjunto de dados que fornece informações sobre as companhias aéreas, presente no *dataset Carriers*, tem a seguinte estrutura, presente na figura 19.

	Code	Description
2	02Q	Titan Airways
3	04Q	Tradewind Aviation
4	05Q	Comlux Aviation, AG
5	06Q	Master Top Linhas Aereas Ltd.
6	07Q	Flair Airlines Ltd.
7	09Q	Swift Air, LLC

Figura 19 - Dataset Carriers.

A descrição dos atributos referentes às companhias aéreas pode ser visualizada na tabela 11.

Tabela 11 - Atributos do dataset Carriers.

Atributo	Exemplo	Descrição do Atributo
Code	02Q	Código da companhia aérea.
Description	Titan Airways	Descrição da companhia aérea.

O conjunto de dados que fornece informações sobre os voos tem a seguinte estrutura, presente nas figuras 20 e 21.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Year	Month	DayofMor	DayOfWe	DepTime	CRSDepTi	ArrTime	CRSArrTin	UniqueCa	FlightNun	TailNum	ActualEla	CRSElapse	AirTime	ArrDelay
2	1996	1	29	1	2039	1930	2245	2139	DL	345	N673DL	246	249	230	66
3	1996	1	30	2	1931	1930	2142	2139	DL	345	N686DA	251	249	224	3
4	1996	1	31	3	1956	1930	2231	2139	DL	345	N685DA	275	249	241	52
5	1996	1	1	1	1730	1550	1909	1745	DL	411	N522DA	219	235	201	84

Figura 20 - Dataset Flights - parte 1.

P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
DepDelay	Origin	Dest	Distance	TaxiIn	TaxiOut	Cancelled	Cancellati	Diverted	CarrierDel	WeatherD	NASDelay	SecurityD	LateAircraftDelay
69	ATL	PHX	1587	6	10	0	NA	0	NA	NA	NA	NA	NA
1	ATL	PHX	1587	5	22	0	NA	0	NA	NA	NA	NA	NA
26	ATL	PHX	1587	7	27	0	NA	0	NA	NA	NA	NA	NA
100	ATL	PHX	1587	4	14	0	NA	0	NA	NA	NA	NA	NA

Figura 21 - Dataset Flights - parte 2.

A descrição dos atributos referentes aos voos pode ser visualizada na tabela 12.

Tabela 12 - Atributos do dataset Flights.

Atributo	Exemplo	Descrição do Atributo
Year	2008	Ano do voo.
Month	1	Mês do voo.
DayofMonth	2	Dia do Mês do voo.
DayOfWeek	3	Dia da semana do voo.
DepTime	2003	Tempo actual de partida.
CRSDepTime	1955	Tempo de partida programado.
ArrTime	2211	Tempo atual de chegada.
CRSArrTime	2225	Tempo de chegada programado.
UniqueCarrier	WN	Código da companhia aérea.
FlightNum	335	Número do voo.
TailNum	N712SW	Identificação do avião.
ActualElapsedTime	128	Tempo atual do voo.
CRSElapsedTime	150	Tempo programado do voo.
AirTime	116	Tempo real de voo.
ArrDelay	-14	Tempo de atraso na chegada.
DepDelay	8	Tempo de atraso na partida.
Origin	IAD	Código IATA do aeroporto de partida.
Dest	TPA	Código IATA do aeroporto de chegada.
Distance	810	Distância percorrida do voo.

Atributo	Exemplo	Descrição do Atributo
TaxiIn	4	Tempo de táxi de chegada.
TaxiOut	8	Tempo de táxi de saída.
Cancelled	0	O voo foi cancelado? 0 se não, 1 se sim.
CancellationCode	*NULL*	Código do Cancelamento.
Diverted	0	O voo foi desviado? 0 se não, 1 se sim.
CarrierDelay	NA	Atraso da operadora.
WeatherDelay	NA	Atraso por maus tempo.
NASDelay	NA	Atraso imposto pelo Sistema nacional do espaço aéreo.
SecurityDelay	NA	Atraso na segurança.
LateAircraftDelay	NA	Atraso no voo anterior.

Uma vez conhecidos os dados para este projeto de dissertação, é fundamental realizar uma limpeza e tratamento aos mesmos, de modo a mitigar qualquer incongruência ou inconsistência nos dados. Todos os dados foram carregados através da ferramenta "*import*" do *SQL Server 2014* para uma base de dados temporária onde foi realizado o tratamento de dados. Existiram 4 tabelas temporárias: *Flight*, *Airport*, *Aiplanes* e *Carriers*, onde o seu principal objetivo foi armazenar e juntar os dados numa só base de dados relacional para que o seu tratamento fosse mais fácil de ser realizado. Na figura 22 podemos visualizar um exemplo do carregamento dos dados referentes aos voos de 1995 para a tabela temporária designada de "*Flight*". O mesmo procedimento foi realizado para todos os outros voos dos restantes anos.

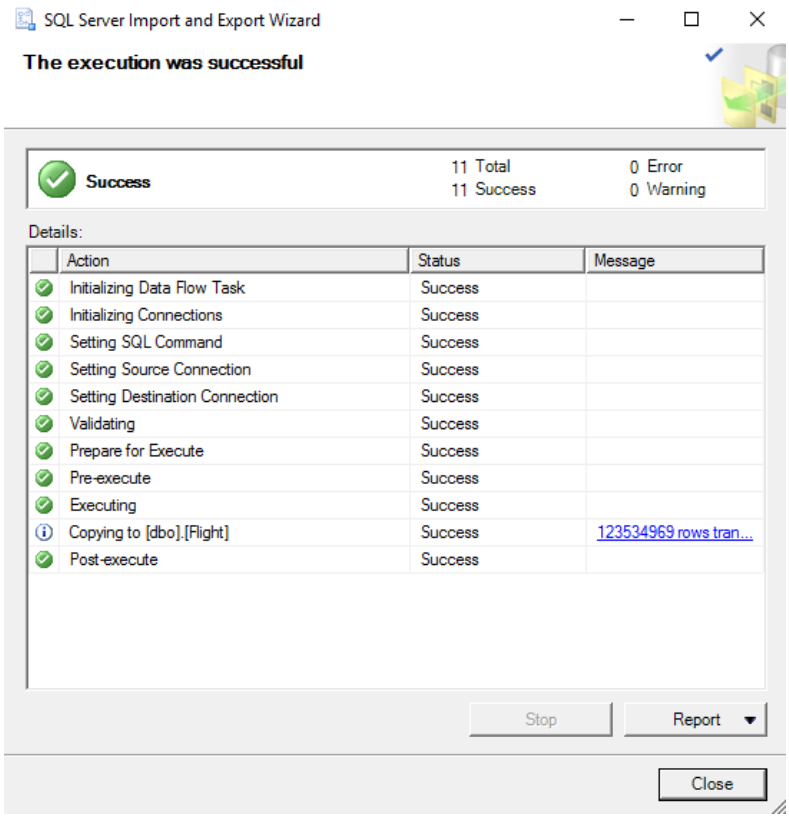

Figura 22 - Importação do dataset Flight para a base de dados relacional temporária.

Antes do tratamento dos dados, os *datasets* referentes aos anos de 1987 até 1994, inclusive, foram descartados por não apresentarem dados que tornam a sua modelação em grafo impossível de ser realizada. Como se pode visualizar nas figuras 23 e 24, encontra-se um exemplo em que os campos referentes à identificação do avião (*TailNum*) se encontra a "NA", o que torna impossível a relação deste voo com o avião associado a esse voo que se encontra na tabela *Plane-data* e que foi posteriormente armazenado na base de dados relacional *airplanes*. No ponto *4.5.1*, é justificado a importância desse e de outros atributos na modelação dos dados em grafos.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	FlightNum	TailNum	ActualElapsedTime	CRSElapsedTime	AirTime	ArrDelay
2	1993	1	29	5	1055	1055	1228	1212	US	6	NA	93	77	NA	16
3	1993	1	30	6	1052	1055	1214	1212	US	6	NA	82	77	NA	2
4	1993	1	31	7	1103	1055	1213	1212	US	6	NA	70	77	NA	1
5	1993	1	1	5	1738	1730	1949	1941	US	6	NA	131	131	NA	8

Figura 23 - Exemplo de anomalias no dataset Flights - Parte 1.

P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
DepDelay	Origin	Dest	Distance	TaxiIn	TaxiOut	Cancelled	CancellationCode	Diverted	CarrierDelay	WeatherDelay	NASDelay	SecurityDelay	LateAircraftDelay
0	EWR	BUF	282	NA	NA	0	NA	0	NA	NA	NA	NA	NA
-3	EWR	BUF	282	NA	NA	0	NA	0	NA	NA	NA	NA	NA
8	EWR	BUF	282	NA	NA	0	NA	0	NA	NA	NA	NA	NA
8	MCO	PIT	834	NA	NA	0	NA	0	NA	NA	NA	NA	NA

Figura 24 - Exemplo de anomalias no dataset Flights - Parte 2.

Todas as manipulações de dados foram realizadas através de *scripts* em *SQL*.

Na tabela "*Flight*", foi adicionada uma nova coluna, designada como "*idFlight*", que identifica cada voo realizado. Este *ID* foi criado pois nos dados fornecidos não há nenhum atributo que distinga cada voo ao longo dos anos.

As colunas *LateAircraftDelay*, *SecurityDelay*, *NASDelay*, *WeatherDelay*, *CarrierDelay*, *CancellationCode*, *Diverted* e *Cancellation* foram removidas por apresentarem todos os registos a "NA", ou em branco, não fornecendo informação útil para o projeto, apenas ocupando espaço desnecessário, que neste caso era essencial, uma vez que a máquina onde estão armazenados os dados, não tem mais memória disponível.

Todos os voos que foram cancelados e divergidos foram também removidos por não permitirem a modelação dos dados e consequente criação de relacionamentos no modelo em grafos. Todos os campos que continham "*NA*" ou "*Unknown*" foram também removidos pela mesma razão anteriormente referida, ou seja, não fornecem informação útil, ocupando espaço desnecessário. As figuras 25, 26 e 27 contêm *scripts* que demonstram como os dados foram tratados.

```
DELETE
FROM [FlightTest].[dbo].[Flight]
WHERE [Diverted]=1 or [Cancelled] = 1
```

Figura 25 - Script para apagar voos cancelados ou desviados.

```
DELETE
FROM [dbo].[Flight]
WHERE [Year] = 'NA' or
[Month] = 'NA' or [DayofMonth] = 'NA' or [DayOfWeek] = 'NA' or [DepTime] =
'NA' or [CRSDepTime] = 'NA' or [ArrTime] = 'NA' or [CRSArrTime] = 'NA' or
[UniqueCarrier] = 'NA' or [FlightNum] = 'NA' or [TailNum] = 'NA' or
[ActualElapsedTime] = 'NA' or [CRSElapsedTime] = 'NA' or [AirTime] = 'NA' or
[ArrDelay] = 'NA' or [DepDelay] = 'NA' or [Origin] = 'NA' or [Dest] = 'NA' or
[Distance] = 'NA' or [TaxiIn] = 'NA' or [TaxiOut] = 'NA' or [Cancelled] = 'NA'
or [Diverted] = 'NA'
```

Figura 26 - Script para apagar valores a "NA" da tabela Flight.

No que toca à tabela dos aeroportos, as colunas referentes à latitude (*Lat*) e longitude (*Long*) foram também removidas pois muitos registos continham os campos em branco, ocupando espaço em disco desnecessário. Na primeira coluna desta mesma tabela (tabela *Airport*), foram também removidas as aspas (“”) que estavam presentes em todos os registos. Esta transformação de dados deve-se ao facto de na tabela *Flight,* os atributos *dest* e *origin* que se relacionam com a tabela *Airport,* os códigos surgirem com o mesmo valor, mas sem aspas. Uma vez removidas, uma ligação entre as duas tabelas já era passível de ser realizada.

```
DELETE
FROM [FlightTest].[dbo].[Flight]
WHERE [Year] = 'UNKNOWN' or [Month] = 'UNKNOWN' or [DayofMonth] = 'UNKNOWN' or
[DayOfWeek] = 'UNKNOWN' or [DepTime] = 'UNKNOWN' or [CRSDepTime] = 'UNKNOWN'
or [ArrTime] = 'UNKNOWN' or [CRSArrTime] = 'UNKNOWN' or [UniqueCarrier] =
'UNKNOWN' or [FlightNum] = 'UNKNOWN' or [TailNum] = 'UNKNOWN' or
[ActualElapsedTime] = 'UNKNOWN' or [CRSElapsedTime] = 'UNKNOWN' or [AirTime] =
'UNKNOWN' or [ArrDelay] ='UNKNOWN' or [DepDelay] = 'UNKNOWN' or [Origin] =
'UNKNOWN' or [Dest] = 'UNKNOWN' or [Distance] = 'UNKNOWN' or [TaxiIn] =
'UNKNOWN' or [TaxiOut] = 'UNKNOWN' or [Cancelled] = 'UNKNOWN' or [Diverted] =
'UNKNOWN'
```

Figura 27 - Script para apagar valores a "Unknown" da tabela Flight.

Nas seguintes figuras podemos visualizar como a coluna “*iata*” se encontrava na tabela *Airport* (figura 28), como se encontrava na tabela *Flight,* (que corresponde às colunas *Origin* e *Dest*) (figura 29), o *script* utilizado para a correção (figura 30) e o resultado final (figura 31).

	iata	airport	city	state
1	"00M"	"Thigpen "	"Bay Springs"	"MS"
2	"00R"	"Livingston Municipal"	"Livingston"	"TX"
3	"00V"	"Meadow Lake"	"Colorado Springs"	"CO"
4	"01G"	"Perry-Warsaw"	"Perry"	"NY"
5	"01J"	"Hilliard Airpark"	"Hilliard"	"FL"
6	"01M"	"Tishomingo County"	"Belmont"	"MS"
7	"02A"	"Gragg-Wade "	"Clanton"	"AL"
8	"02C"	"Capitol"	"Brookfield"	"WI"
9	"02G"	"Columbiana County"	"East Liverpool"	"OH"

Figura 28 - Tabela Airport antes do tratamento de dados.

idFlight	year	month	dayOfMonth	dayOfWeek	depTime	crsDepTime	arrTime	crsArrTime	flightNum	actualElapsedTime	crsElapsedTime	airTime	arrDelay	depDelay	distance	taxiIn	taxiOut	dest	origin	ta
1	1995	1	6	5	657	645	952	937	482	115	112	83	15	12	678	7	25	PHL	ORD	N
2	1995	1	7	6	648	645	938	937	482	110	112	88	1	3	678	5	17	PHL	ORD	N
3	1995	1	8	7	649	645	932	937	482	103	112	83	-5	4	678	3	17	PHL	ORD	N
4	1995	1	9	1	645	645	928	937	482	103	112	84	-9	0	678	3	16	PHL	ORD	N

Figura 29 - Tabela Flight.

```
UPDATE [dbo].[Airport]
SET [iata] = replace([iata],'"','')
```

Figura 30 - Script para remoção das aspas ao atributo Iata da tabela Airport.

	iata	airport	city	state	country
1	00M	"Thigpen "	"Bay Springs"	"MS"	"USA",31.95376472,-89.23450472
2	00R	"Livingston Municipal"	"Livingston"	"TX"	"USA",30.68586111,-95.01792778
3	00V	"Meadow Lake"	"Colorado Springs"	"CO"	"USA",38.94574889,-104.5698933
4	01G	"Perry-Warsaw"	"Perry"	"NY"	"USA",42.74134667,-78.05208056
5	01J	"Hilliard Airpark"	"Hilliard"	"FL"	"USA",30.6880125,-81.90594389
6	01M	"Tishomingo County"	"Belmont"	"MS"	"USA",34.49166667,-88.20111111
7	02A	"Gragg-Wade "	"Clanton"	"AL"	"USA",32.85048667,-86.61145333
8	02C	"Capitol"	"Brookfield"	"WI"	"USA",43.08751,-88.17786917
9	02G	"Columbiana County"	"East Liverpool"	"OH"	"USA",40.67331278,-80.64140639
10	03D	"Memphis Memorial"	"Memphis"	"MO"	"USA",40.44725889,-92.22696056

Figura 31 - Tabela Airport depois do tratamento da coluna Iata.

O mesmo procedimento foi realizado para a tabela *Airline*, retirando as aspas. O código e o resultado final podem ser visualizados nas figuras 32 e 33 respectivamente.

```
UPDATE [dbo].[Airline]
SET [idAirline] = replace([idAirline], '"', '')
```

Figura 32 - Script para remoção das aspas ao atributo idAirline da tabela Airline.

	idAirline	description
1	02Q	"Titan Airways"
2	04Q	"Tradewind Aviation"
3	05Q	"Comlux Aviation, AG"
4	06Q	"Master Top Linhas Aereas Ltd."
5	07Q	"Flair Airlines Ltd."
6	09Q	"Swift Air, LLC"
7	0BQ	"DCA"
8	0CQ	"ACM AIR CHARTER GmbH"
9	0FQ	"Maine Aviation Aircraft Charter, LLC"
10	0GQ	"Inter Island Airways, d/b/a Inter Island Air"
11	0HQ	"Polar Airlines de Mexico d/b/a Nova Air"

Figura 33 - Tabela Airline depois do tratamento realizado.

O resultado final, no que toca à quantidade de dados, depois da limpeza e tratamento de dados encontra-se resumido na tabela 13.

Tabela 13 - Quantidade de registos nas tabelas após o tratamento aos dados.

Dataset	Quantidade de registos
1995	5 219 140
1996	5 209 326
1997	5 301 999
1998	5 227 051
1999	5 360 018
2000	5 481 303
2001	5 723 673
2002	5 197 739
2003	6 375 689
2004	6 987 729
2005	6 992 838
2006	7 003 802
2007	7 275 288
2008	6 855 029
airplane	5 030
airline	1 492
airports	3 377
Total	84 220 523

4.3 Carregamento da Base de Dados Relacional

O SGBD relacional escolhido para este projeto de dissertação foi o *SQL Server* 2014, onde foi realizado todo o tratamento dos dados, assim como o armazenamento dos diversos cenários das bases de dados relacionais utilizadas neste projeto de dissertação. Segundo o (DBEngine, 2017), o *SQL Server* foi considerado o SGBDR do ano por ter sido o que ganhou mais popularidade durante o ano de 2016 de entre 315 sistemas monitorizados. Esta escolha encontra-se assim, justificada pelo facto da sua margem de crescimento ser maior que a concorrência.

4.3.1 Modelo Conceptual

O modelo conceptual de dados permite uma fácil compreensão de como os dados estão organizados em entidades e como estas estão relacionadas. O modelo relacional deste projeto tem o esquema presente na figura 34.

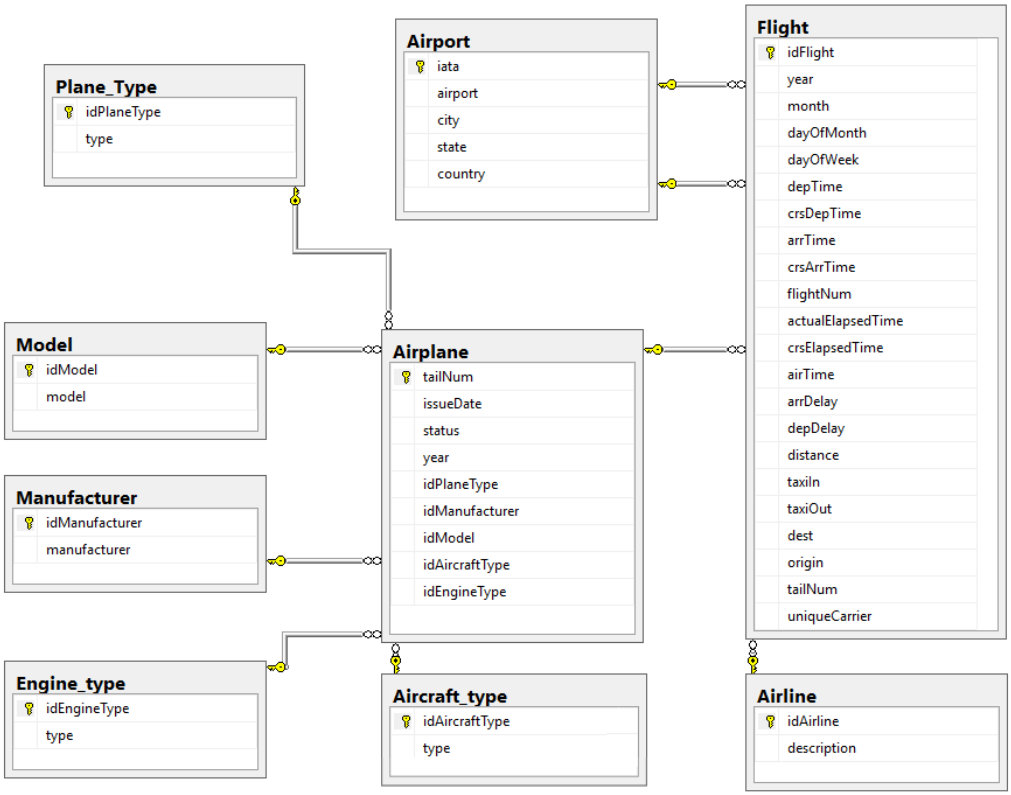

Figura 34 - Modelo conceptual da base de dados relacional.

Como podemos visualizar, a base de dados relacional encontra-se dividida em 9 entidades, sendo descritas no ponto 4.3.2. Neste modelo conceptual da base de dados, destacam-se as tabelas *Airplane* e *Flight*. A tabela *Airplane* encontra-se diretamente relacionada com as tabelas *Aircraft_type*, *Engine_Type*, *Manufacturer*, *Model*, e *Plane_Type*. A tabela *Flight* encontra-se diretamente relacionada com a tabela *Airplane*, *Airline*, e duplamente relacionada com a tabela *Airport*, uma vez que um voo tem sempre, obrigatoriamente, um aeroporto de origem e destino.

O código para a criação da base de dados relacional pode ser encontrado nos Anexos.

4.3.2 Descrição das Entidades

Flight

Esta entidade tem como objetivo armazenar dados sobre cada voo realizado (tabela 14).

Tabela 14 - Entidade Flight da base de dados relacional.

Atributo	Descrição
idFlight	Identificador único de cada voo. É o único atributo que distingue os voos, sendo por essa razão a chave primária da entidade *Flight*.
year	Indica o ano do voo.
month	Indica o mês do voo.
dayOfMonth	Indica o dia do mês do voo.
dayOfWeek	Indica o dia da semana (de 1 a 7) do voo.
depTime	Indica o tempo real da partida do voo.
crsDepTime	Indica o tempo programado para a partida do voo.
arrTime	Indica o tempo real da chegada do voo.
crsArrTime	Indica o tempo programado para a chegada do voo.
flightNum	Indica o número do voo.
actualElapsedTime	Indica o tempo real do voo.
crsElapsedTime	Indica o tempo programado do voo.
airTime	Indica o tempo em que o voo esteve no ar.
arrDelay	Indica o tempo de atraso na chegada.
depDelay	Indica o tempo de atraso na partida.
Distance	Indica a distância percorrida do voo.
taxiIn	Indica o tempo de táxi na chegada.
taxiOut	Indica o tempo de táxi na partida.
Dest	Indica o aeroporto de destino. Esta é uma chave estrangeira que faz a ligação com a tabela *Airport*.
Origin	Indica o aeroporto de partida. Esta é uma chave estrangeira que faz a ligação com a tabela *Airport*.
tailNum	Indica a matrícula ou identificação do avião. Esta é uma chave estrangeira que faz a ligação com a tabela *Airplane*. Tem uma ligação de muitos para um, ou seja, um voo tem um avião associado, que é identificado pelo *tailNum*. Um avião pode realizar muitos voos.
uniqueCarrier	Indica a companhia aérea responsável pelo voo. Esta é uma chave estrangeira que faz a ligação com a tabela *Airline*. Tem uma ligação de muitos para um, ou seja, um voo tem uma companhia aérea

Atributo	Descrição
	associada, por outro lado, uma companhia aérea pode realizar muitos voos.

Airline

Esta entidade tem como objetivo armazenar dados sobre cada companhia aérea que realiza o voo (tabela 15).

Tabela 15 - Entidade Airline da base de dados relacional.

Atributo	Descrição
idAirline	Identificador único para as companhias aéreas. Esta é uma chave primária da entidade *Airline* e está relacionada diretamente com a tabela *Flight*.
description	Indica o nome ou designação da companhia aérea.

Airplane

Esta entidade tem como objetivo armazenar dados sobre o avião que realiza o voo (tabela 16).

Tabela 16 - Entidade Airplane da base de dados relacional.

Atributo	Descrição
tailNum	Identificador único para o avião do voo. Esta é uma chave primária da entidade *Airplane* e está relacionada diretamente com a tabela *Flight*.
issueDate	Indica a data de registo do avião.
status	Indica o estado atual do avião, se se encontra válido ou não.
year	Indica o ano de fabrico do avião.
idPlaneType	Indica o tipo de voo que o avião está a realizar. Esta é uma chave estrangeira que faz a ligação com a tabela *Plane_Type*. Tem uma relação de um para muitos, ou seja, cada avião tem um tipo de avião ou de voo, enquanto que cada tipo de voo pode ser realizado por muitos aviões.
idManufacturer	Indica o construtor ou marca do avião. Esta é uma chave estrangeira que faz a ligação com a tabela *Manufacturer*. Tem uma relação de um para muitos, ou seja, cada avião tem um fabricante ou *Manufacturer*, enquanto que cada fabricante tem muitos aviões.
idModel	Indica o modelo do avião que está a realizar o voo. Esta é uma chave estrangeira que faz a ligação com a tabela *Model*. Tem uma relação

Atributo	Descrição
	de um para muitos, ou seja, cada avião tem um modelo, enquanto que cada modelo de avião pode ter muitos aviões.
idAircraftType	Indica o tipo de avião que está a realizar o voo. Esta é uma chave estrangeira que faz a ligação com a tabela *Aircraft_Type*. Tem uma relação de um para muitos, ou seja, cada avião tem um tipo de avião, enquanto que cada tipo de avião pode ter muitos aviões.
idEngineType	Indica o tipo de motor do avião que está a realizar o voo. Esta é uma chave estrangeira que faz a ligação com a tabela *Engine_Type*. Tem uma relação de um para muitos, ou seja, cada avião tem um tipo de motor, enquanto que cada tipo de motor pode ter muitos aviões.

Airport

Esta entidade tem como objetivo armazenar dados sobre os aeroportos, que podem ser de chegada e destino (tabela 17). Tem duas ligações de um para muitos com a tabela *Flight*, devido ao facto de um voo ter sempre um aeroporto de destino e de origem. Cada um desses aeroportos pode ter um ou mais voos.

Tabela 17 - Entidade Airport da base de dados relacional.

Atributo	Descrição
iata	Identificador único para um aeroporto. Esta é uma chave primária da entidade *Airport* e está relacionado duplamente com a tabela *Flight*, pois por cada voo existe um aeroporto de origem e de destino, resultando daí as duas relações de um para muitos.
airport	Indica a designação oficial do aeroporto.
city	Indica a cidade onde se localiza o aeroporto.
state	Indica o estado onde se localiza o aeroporto.
country	Indica o país onde se localiza o aeroporto.

Aircraft_type

Esta entidade tem como objetivo armazenar dados sobre o tipo de aviões que realizaram os voos (tabela 18).

Tabela 18 - Entidade Aircraft_type da base de dados relacional.

Atributo	Descrição
idAircraftType	Identificador único para o tipo de avião que efetua o voo. Esta é uma chave primária da entidade *Aircraft_Type* e está relacionado com a

Atributo	Descrição
	tabela *Airplane*, pois cada avião pertence a um tipo ou categoria de avião.
type	Indica a designação de cada tipo de avião.

Engine_type

Esta entidade tem como objetivo armazenar dados sobre o tipo de motor dos aviões que realizaram os voos (tabela 19).

Tabela 19 - Entidade Engine_Type da base de dados relacional.

Atributo	Descrição
idEngineType	Identificador único para o motor do avião. Esta é uma chave primária da entidade Engine_*Type* e está relacionado com a tabela *Airplane*, pois cada avião tem um tipo de motor.
type	Indica a designação para cada tipo de motor do avião.

Manufacturer

Esta entidade tem como objetivo armazenar dados sobre o fabricante dos aviões que realizaram os voos (tabela 20).

Tabela 20 - Entidade Manufacturer da base de dados relacional.

Atributo	Descrição
idManufacturer	Identificador único para um construtor ou fabricante do avião. Esta é uma chave primária da entidade *Manufacturer* e está relacionado com a tabela *Airplane*, pois cada avião tem obrigatoriamente um fabricante.
manufacturer	Indica a designação para cada fabricante do avião.

Model

Esta entidade tem como objetivo armazenar dados sobre o modelo dos aviões que realizaram os voos (tabela 21).

Tabela 21 - Entidade Model da base de dados relacional.

Atributo	Descrição
idModel	Identificador único para o modelo do avião. Esta é uma chave primária da entidade *Model* e está relacionado com a tabela

Atributo	Descrição
	Airplane, pois cada avião tem obrigatoriamente um modelo associado.
model	Indica a designação do modelo do avião.

Plane_type

Esta entidade tem como objetivo armazenar dados sobre o tipo de voo que o avião estava a realizar (tabela 22).

Tabela 22 - Entidade Plane_Type da base de dados relacional.

Atributo	Descrição
idPlaneType	Identificador único que distingue o tipo de avião. Esta é uma chave primária da entidade *Plane_Type* e está relacionado com a tabela *Airplane*, pois cada avião tem obrigatoriamente um tipo de voo associado.
type	Indica a designação de cada tipo de avião.

4.4 Carregamento da Base de Dados Relacional

Todo o carregamento da base de dados relacional foi realizado através de scripts em *SQL*.

Como explicado no ponto *4.2*, foram criadas 4 tabelas temporárias em *SQL*, onde foi realizado algum tratamento de dados. Uma vez realizada a limpeza e tratamento dos dados, foi criado um modelo relacional, como demonstrado no ponto *4.3.1*. A tabela temporária *Flights* foi ainda dividida em 4 outras tabelas de acordo com os cenários criados para conseguir diferentes volumes de bases de dados. O primeiro cenário contém voos apenas do ano 1995, o segundo cenário contém os voos de 1995 a 1999, o terceiro cenário contém os voos de 1995 a 2004, e o último cenário contém os voos de 1995 a 2008. Destas tabelas auxiliares foi possível realizar o carregamento para as tabelas do modelo relacional de acordo com os diferentes cenários das bases de dados.

Foi primeiramente efetuado o carregamento das tabelas *Airport, Airline, Aircraft_Type, Engine_Type, Manufacturer, Model, e Plane_Type*, pois estas não tinham dependências de outras tabelas. O *script* presente na figura 35, foi utilizado para carregar os dados da tabela auxiliar *Airport* para a tabela *Airport* do cenário 4 (*Schema Scn4*).

```
INSERT INTO [scn4].[dbo].[Airport] (iata, airport, city, [state], country)
SELECT iata, airport, city, [state], country
FROM [Dissertacao].[dbo].[Airport]
```

Figura 35 - Script de carregamento da tabela Airport do cenário 4.

Os restantes *scripts* para o carregamento das tabelas *Airline, Aircraft_Type, Engine_Type, Manufacturer, Model,* e *Plane_Type,* que são em tudo parecidas ao script da figura 35, podem ser encontrados em Anexo.

O carregamento das tabelas *Airplane* e *Flight* é mais complexo, uma vez que têm dependências que são chaves principais em outras tabelas, sendo assim importante garantir que apenas são inseridos dados que já existam noutras tabelas secundárias. O *script* utilizado para o carregamento da tabela *Airplane* pode ser visualizado na figura 36.

```
INSERT INTO [scn4].[dbo].Airplane(tailNum, issueDate, [status], [year],
idPlaneType, idManufacturer, idModel, idAircraftType, idEngineType)
(SELECT distinct tailNum, issueDate, [status], [year], Pl.idPlaneType,
M.idManufacturer, Mo.idModel, At.idAircraftType, E.idEngineType
       FROM [Dissertacao].[dbo].[Airplane] A
       INNER JOIN [scn4].[dbo].[Plane_Type] Pl
       ON A.[plane_type] = Pl.[type]
       INNER JOIN [scn4].[dbo].[Manufacturer] M
       ON A.[manufacturer] = M.[manufacturer]
       INNER JOIN [scn4].[dbo].[Model] Mo
       ON A.[model] = Mo.[Model]
       INNER JOIN [scn4].[dbo].[Aircraft_type] At
       ON A.[aircraft_type] = At.[type]
       INNER JOIN [scn4].[dbo].[Engine_type] E
       ON A.[engine_type] = E.[type])
```

Figura 36 - Script de carregamento da tabela Airplane.

Para o carregamento da tabela *Flight* foi utilizada a mesma técnica que para a tabela *Airplane*. O *script* utilizado para o carregamento da tabela *Flight* pode ser visualizado na figura 37.

```
INSERT INTO [scn4].[dbo].Flight (idFlight, [year], [month], [dayOfMonth],
[dayOfWeek], depTime, crsDepTime, arrTime, crsArrTime, flightNum,
actualElapsedTime, crsElapsedTime, airTime, arrDelay, depDelay, distance, taxiIn,
taxiOut, dest, origin, tailNum, uniqueCarrier)
       (SELECT DISTINCT [idFlight],NULL, [month], [dayOfMonth], [dayOfWeek],
depTime, crsDepTime, arrTime, crsArrTime, flightNum, actualElapsedTime,
crsElapsedTime, airTime, arrDelay, depDelay, distance, taxiIn, taxiOut, Ap1.iata,
Ap2.iata, A.tailNum, Al.idAirline
       FROM [Dissertacao].[dbo].[flights] F
       INNER JOIN [scn4].[dbo].[Airport] Ap1
       ON F.dest = Ap1.iata
       INNER JOIN [scn4].[dbo].[Airport] Ap2
       ON F.origin = Ap2.iata
       INNER JOIN [scn4].[dbo].[Airplane] A
       ON F.tailNum = A.tailNum
       INNER JOIN [scn4].[dbo].[Airline] Al
       ON F.uniqueCarrier = Al.idAirline)
       WHERE [year] IN ('1995','1996','1997','1998','1999')
```

Figura 37 - Script de carregamento da tabela Flight.

Desde modo, os quatro cenários da base de dados relacional ficam preparados para os testes de desempenho.

4.5 Carregamento da Base de Dados em Grafo

O SGBD em grafo escolhido para este projeto de dissertação foi o *Neo4J Community Edition 3.2.0*, que foi utilizado para a modelação e armazenamento dos dados em grafo. A opção por este SGBD deveu-se ao facto de ser atualmente o maior representante deste tipo de base de dados (DBEngine, 2017) tendo assim uma maior comunidade de suporte e mais informação presente na *web*.

4.5.1 Modelo em Grafo

O modelo em grafo permite representar informação em nodos assim como representar relações com diversas propriedades entre os nodos.

No nosso modelo em grafo foi decidido representar 4 tipos diferentes de nodos, sendo eles *Airline*, *Airplane*, *Airport* e *Flight*. No que toca às relações, ficou definido existirem 4 tipos de relações (*By*, *On*, *From* e *To*).

A relação "*By*" relaciona um voo (*Flight*) a uma companhia aérea (*Airline*), simbolizando que cada voo é realizado por uma companhia aérea. A relação "*On*" relaciona um voo (*Flight*) a um avião (*Airplane*), simbolizando que cada voo é realizado em um avião. Existem duas relações entre um voo (*Flight*) e um

aeroporto (*Airport*), a relação "*From*" e a relação "*To*". A relação "*From*" representa todos os voos que saem de um aeroporto, enquanto a relação "*To*" representa a relação entre um voo e o aeroporto de destino.

O modelo da base de dados em grafo pode ser visualizado na figura 38.

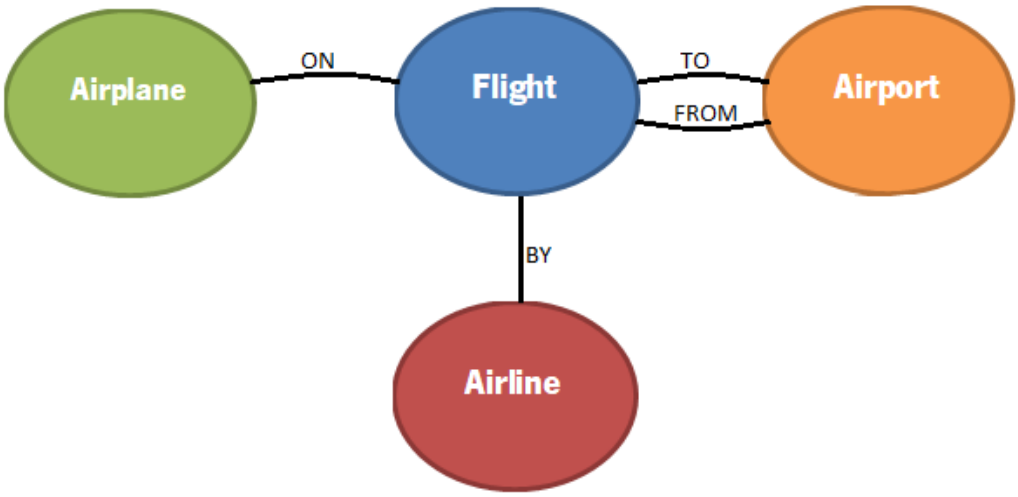

Figura 38 - Modelo da base de dados em grafo.

A construção do modelo em grafo foi realizado a partir do modelo relacional e seguiu algumas *guidelines* definidas por (Hunger, 2016) e (Virgilio, Maccioni, & Torlone, 2013) e consistem no seguinte:

- Relações simples entre duas tabelas, podem tornar-se no modelo em grafo num só nodo. No nosso caso, no modelo relacional, os dados presentes nas entidades *Plane_Type*, *Model*, *Manufacturer*, *Engine_type*, *Aircraft_type*, e *Airplane* do modelo relacional encontram-se num só nodo no modelo em grafo, o nodo *Airplane*;
- Localizar as chaves estrangeiras do modelo relacional, sendo que estas podem passar a ser relações no modelo em grafo. No nosso modelo relacional, as ligações entre as tabelas *Airplane* e *Flight* passaram a ser uma relação no modelo em grafo (a relação "*On*" entre os nodos *Airplane* e *Flight*); As ligações entre as tabelas *Airline* e *Flight* passaram a ser uma relação no modelo em grafo (a relação "*By*" entre os nodos *Airline* e *Flight*); As duplas ligações entre as tabelas *Airport* e *Flight* passaram a ser duas relações no modelo em grafo (as relações "*From*" e "*To*" entre os nodos *Flight* e *Airport*);
- Em relações de N:N no modelo relacional terá de existir uma tabela de ligação. Os atributos dessa tabela de ligação serão as propriedades de uma relação no modelo em grafo. No nosso modelo

relacional não existe este tipo de tabelas, pelo que no modelo em grafo não existem propriedades de relações.

As propriedades em cada nodo são:

Nodo *Flight*:

Tabela 23 - Propriedade do nodo Flight da base de dados em grafo.

Propriedade	Descrição
idFlight	Identificador único que distingue cada voo realizado.
year	Indica o ano do voo.
month	Indica o mês do voo.
dayOfMonth	Indica o dia do mês do voo.
dayOfWeek	Indica o dia da semana do voo.
depTime	Indica o tempo real da partida do voo.
crsDepTime	Indica o tempo programado para a partida do voo.
arrTime	Indica o tempo real da chegada do voo.
crsArrTime	Indica o tempo programado para a chegada do voo.
flightNum	Indica o número do voo.
actualElapsedTime	Indica o tempo real do voo.
crsElapsedTime	Indica o tempo programado para o voo.
airTime	Indica o tempo em que o voo esteve no ar.
arrDelay	Indica o tempo de atraso na chegada.
depDelay	Indica o tempo de atraso na partida.
distance	Indica a distância percorrida do voo.
taxiIn	Indica o tempo de táxi na chegada.
taxiOut	Indica o tempo de táxi na saída.
dest	Indica o aeroporto de destino. É a propriedade que faz ligação com o nodo *Airport* através da relação "*To*".
origin	Indica o aeroporto de origem. É a propriedade que faz ligação com o nodo *Airport* através da relação "*From*".
tailNum	Indica a matricula do avião que efetuou o voo. É a propriedade que faz ligação com o nodo *Airplane* através da relação "*Ori*".
uniqueCarrier	Indica a companhia aérea pela qual o voo foi realizado. É a propriedade que faz ligação com o nodo *Airline* através da relação "*By*".

Nodo *Airport*:

Tabela 24 - Propriedades do nodo Airport da base de dados em grafo.

Propriedade	Descrição
iata	Identificador único que distingue cada aeroporto. É a propriedade do nodo *Airport* que se relaciona com o nodo *Flight*, proporcionando assim as relações "*From*" e "*To*".
country	Indica o país do aeroporto.
city	Indica a cidade do aeroporto.
state	Indica o estado do aeroporto.
Airport	Indica a designação oficial do aeroporto.

Nodo *Airline*:

Tabela 25 - Propriedades do nodo Airline da Base de Dados em Grafos.

Propriedade	Descrição
idAirline	Identificador único para as companhias aéreas. É a propriedade do nodo *Airline* que se relaciona diretamente com o nodo *Flight*, proporcionando assim a relação "*By*".
description	Indica a designação oficial da companhia aérea.

Nodo *Airplane*:

Tabela 26 - Propriedades do nodo Airplane da Base de Dados em Grafo.

Propriedade	Descrição
tailNum	Identificador único para o avião que realizou o voo. É a propriedade do nodo *Airplane* que se relaciona diretamente com o nodo *Flight*, proporcionando assim a relação "*On*".
type	Indica o tipo de voo que o avião está a realizar.
manufacturer	Indica o construtor ou marca do avião.
issue_date	Indica a data de registo do avião.
status	Indica o estado atual do avião.
aircraft_type	Indica o tipo do avião que está a realizar o voo.
engine_type	Indica o tipo do motor do avião que está a realizar o voo.
year	Indica o ano de fabrico do avião.

4.5.2 Extração e Carregamento da Base de Dados em Grafos

Cada cenário tem um carregamento em tudo igual, sendo que apenas varia na quantidade de dados carregados, sendo o único aspeto que o diferencia dos restantes cenários.

Como o carregamento é realizado através de ficheiros *CSV's*, e como os dados foram tratados no *SQL Server*, o primeiro passo passou por exportar do *SQL Server* esses mesmos dados para ficheiros *CSV*. Sendo assim, foram exportados 4 *CSV's* diferentes referentes aos voos, um por cada cenário. Estes *CSV's* dizem respeitos às tabelas auxiliares que foram descritas no ponto 4.2.

No que toca aos nodos *Airline*, *Airplane* e *Airport*, os dados são exatamente os mesmos nos 4 diferentes cenários, sendo assim, foi exportado diretamente do *SQL Server* os dados correspondentes a essas tabelas. No que toca ao nodo *Flight*, este é o único nodo que varia na quantidade de informação armazenada, uma vez que varia de acordo com o cenário.

A criação e carregamento dos nodos foi realizada através da interface em *browser* do *Neo4J*. A figura 39 demonstra o *script* de como foi efetuado o carregamento do nodo *Airline*. Os restantes *scripts* de criação e carregamento dos nodos, podem ser encontrados em Anexo.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///airline.csv" AS row
CREATE (n:Airline)
SET n = row
```

Figura 39 - Criação e carregamento do nodo Airline na base de dados em grafo.

Como podemos visualizar, a estrutura do nodo, isto é, as propriedades que o nodo vai conter são definidas através das colunas que o *CSV* importa para a base de dados, isto é, as colunas que os *CSV's* têm, são as propriedades que os nodos irão ter. Os dados para serem importados são colocados na pasta "...Documents\Neo4j\default.graphdb\import" em formato *CSV*, sendo que cada linha simboliza um registo e a primeira define a estrutura do nodo. O código *Using Periodic Commit* usa-se pelo facto de o *CSV* ser bastante grande, o que iria sobrecarregar a memória durante o carregamento. Assim, a cada 1000 linhas acontecerá um *commit*. O próximo passo é a criação de índices, que tem como objetivo tornar a consulta de dados mais eficiente. Na figura 40 podemos visualizar um exemplo da atribuição de um índice ao nodo *Flight*, nomeadamente à propriedade *idFlight*.

```
CREATE INDEX ON :Flight(idFlight)
```

Figura 40 - Criação do índice no nodo Flight.

Uma vez carregado os nodos da base de dados, torna-se necessário definir as relações entre os nodos. A figura 41 contém o *script* que permitiu o carregamento da relação "*By*" entre o nodo *Flight* e *Airline*:

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///by.csv" AS row
MATCH (f:Flight),(a:Airline)
WHERE f.idFlight = row.idFlight AND a.idAirline = row.uniqueCarrier
CREATE (f)-[details:By]->(a)
SET details = row
```

Figura 41 - Script de carregamento da relação "By" entre o nodo Airline e Flight.

A relação é carregada através de um ficheiro *CSV*, sendo que este apenas contém duas colunas e estão presentes no *dataset* "*by.csv*". Na figura 41 podemos verificar que o *CSV* apenas contém as propriedades *idFlight* e *uniqueCarrier*, onde a propriedade *idFlight* faz a ligação com a propriedade *idFlight* do nodo *Flight* e a propriedade *uniqueCarrier* faz a ligação com a propriedade *idAirline* do nodo *Airline*. Estes *CSVs* foram criados através da ferramenta de exportação do *SQL Server* onde apenas foram selecionadas duas colunas, que neste caso especifico são a *idFlight* e *uniqueCarrier*. O mesmo processo foi realizado para as restantes relações entre os nodos. Os códigos para o carregamento dessas relações podem ser encontrados em Anexo.

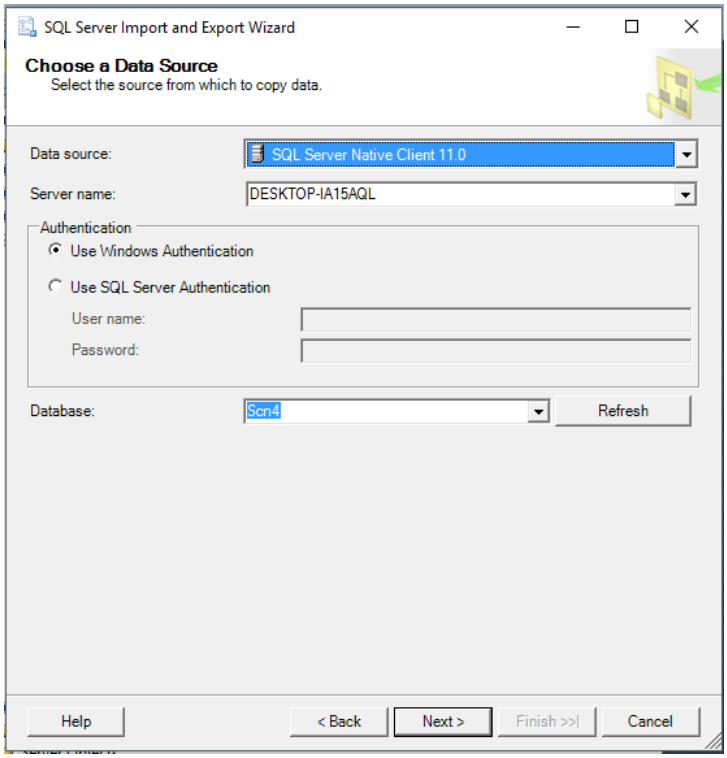

Figura 42 - Exportação de uma tabela de uma relação para um ficheiro CSV.

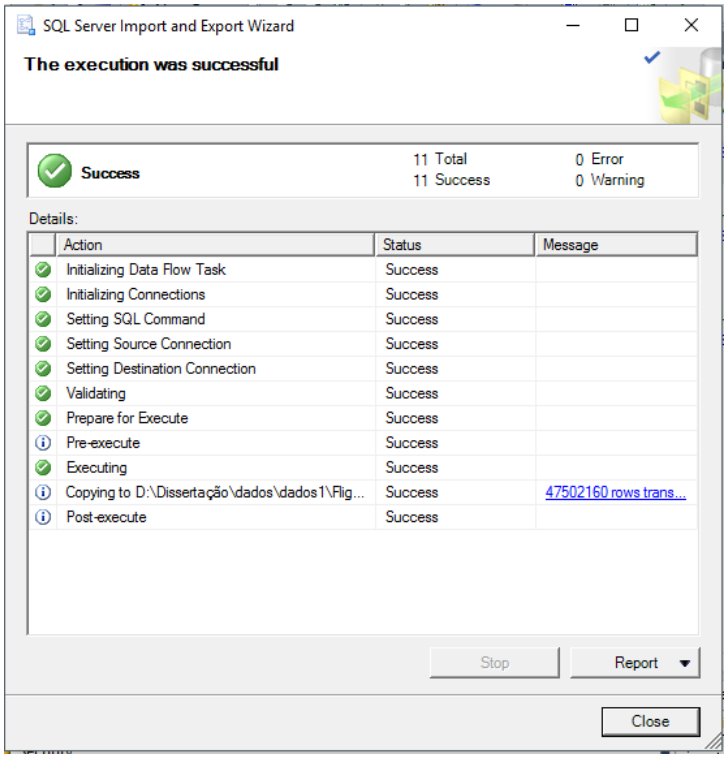

Figura 43 - Exportação realizada com sucesso.

Se o número de linhas transferidas for exatamente o mesmo que o número de voos de cada cenário, podemos confirmar que todos os voos têm todas as relações com os restantes nodos. Nas duas figuras anteriores (figura 42 e 43), podemos verificar a extração de dados para um *CSV*, que irá permitir fazer as relações no *Neo4J*. Cada cenário terá um *CSV* diferente para cada relação, ou seja, há um total de 16 *datasets* para a criação e carregamento das relações (4 relações X 4 cenários = 16 *datasets*).

Este processo de extração de duas colunas foi realizado para as quatro relações. As propriedades associadas a cada relação estão presentes na tabela 27.

Tabela 27 - Propriedades associadas a cada relação.

Relação	Propriedades
By	*idFlight, uniqueCarrier*
On	*idFlight, tailNum*
From	*idFlight, origin*
To	*idFlight, dest*

5. BENCHMARKING

O *Benchmarking* é um processo de comparação de desempenho entre dois produtos. Segundo (Stapenhurst, T., 2009) um *benchmarking* envolve a comparação de métricas de desempenho de um produto, atividades ou processos que são comparadas ente si, tendo como objetivo encontrar o que melhor desempenha tendo em conta as métricas definidas. Segundo (Dragolea L. & Cotirlea, D, 2009), Xerox afirmou que os *benchmarkings* são muito utilizados porque:

- É parte de uma cultura de melhoria;
- Pode ser considerado um atalho para melhorar os processos, uma vez que uma das bases de dados pode não estar a ser utilizada e, no entanto, ter melhor desempenho para certas necessidades;
- É um mecanismo que leva à melhoria constante, pois nenhuma base de dados quer ter um desempenho inferior à concorrência;
- Resolve problemas;
- É um dos requisitos para a excelência nos modelos de negócios;
- Ajuda a justificar novos produtos, processos e práticas organizacionais, onde neste caso pode afirmar a afirmação do *Neo4J* no mercado das bases de dados;
- Ajuda a definir um dos pontos fracos da concorrência, através de resultados menos bons por parte da concorrência.

Neste projeto de dissertação, os produtos *Neo4J Community Edition* e o *SQL Server 2014* são alvo de uma comparação, sendo o principal objetivo determinar qual a tecnologia de base de dados mais indicada para lidar com determinado tipo e dimensão de dados, no que toca ao desempenho através de consultas às bases de dados.

Para a realização deste *benchmarking*, foi utilizado um computador com as características presentes na tabela 28.

Tabela 28 - Características da máquina que realizou o Benchmarking.

Processador	Inter(R) Core™ i7-6700HQ CPU 2.60GHz
Memória RAM	16 GB
Sistema Operativo	Windows 10 Pro x64

5.1 Aplicação para a realização do Benchmarking

Optou-se por desenvolver uma aplicação na linguagem *R* para fazer o teste de desempenho aos dois produtos. O *R* é uma linguagem de programação estatística que tem ganho cada vez mais popularidade entre aquelas de manipulação e tratamento de dados (Braun & Murdoch, 2007). O objetivo inicial era realizar este teste através da aplicação *Apache Jmeter*, contudo, devido a problemas de conectividade entre o *Neo4J* e o *Java,* que não foram passíveis de ser resolvidos, foi criada a aplicação em *R*. O *R* tem tido cada vez mais importância na comunidade dos profissionais de dados, nomeadamente entre os *Data Scientists*, *Business Intelligence Developers* entre muitos outros. O crescimento da sua comunidade de utilizadores tem crescido a uma taxa cada vez maior, razão pela qual esta foi a linguagem escolhida para este projeto de dissertação.

O código desenvolvido para esta aplicação pode ser encontrado em Anexo. A construção desta aplicação, seguiu as mesmas normas, tanto para as consultas à base de dados relacional, como para a base de dados em grafo, isto é, ambos começam por inicializar uma ligação à base de dados, executam a consulta, devolvem o resultado, e fecham a ligação. Se o número de utilizadores for maior que um, irão ser realizados vários ciclos do processo anteriormente descrito de acordo com o número de utilizadores escolhidos. O tempo de execução da consulta, corresponde ao tempo total da execução das instruções, isto é, desde que a conexão é iniciada até ser finalizada.

Durante a realização dos testes, a máquina esteve sempre sujeita à mesma carga, isto é, apenas a aplicação se encontrava em execução, permitindo assim ter consultas com o mesmo nível de igualdade no que toca à memória e recursos disponíveis da máquina, ou seja o ambiente é o mesmo. Deste modo, foi respeitado uma das condições necessárias para a realização de *benchmarkings* definida por (Stapenhurst, T., 2009). Todas as variáveis de ambiente no *R* foram apagadas após cada teste. Desta forma, conseguimos obter uns testes consistentes e livres de qualquer influência externa.

5.1.1 *Benchmarking* à Base de Dados Relacional

A aplicação em *R* que permitiu medir o desempenho das consultas à base de dados relacional foi desenvolvida utilizando a biblioteca "*RODBC*", que é um *driver ODBC* (*Open Database Connectivity*) que permite ligar os sistemas de gestão de base de dados relacionais, neste caso o *SQL Server* 2014, através da linguagem *R*. Utilizando essa mesma biblioteca, foi possível construir a função "*SQLServerTest*" que permite realizar as consultas e medir o seu desempenho.

Nesta aplicação, existem algumas variáveis, presentes na tabela 29, que são fundamentais para a realização dos testes de desempenho:

Tabela 29 - Variáveis fundamentais na consulta à base de dados relacional.

Variável	Exemplo da utilização	Descrição
start.time	start.time <- Sys.time()	A variável *start.time* permite armazenar o exato momento em que o teste começa a ser realizado.
con	con <-odbcConnect("Scn2", uid = "", pwd = "")	A variável *con* permite guardar e iniciar uma conexão com a base de dados relacional através da função *odbcConnect*, sendo que nesta função o primeiro parâmetro diz respeito à origem dos dados (Neste exemplo encontra-se a conexão à base de dados "*Scn2*"), o segundo parâmetro informa sobre a identificação do utilizador que vai aceder à base de dados e o terceiro parâmetro informa sobre a palavra-passe desse mesmo utilizador.
SQLQuery	SQLQuery <- "SELECT * FROM Flight"	A variável *SQLQuery* contém o *script* da consulta à base de dados relacional. São utilizadas nesta consulta as *queries* definidas no ponto 5.3.
SQLcommand	SQLcommand <- RODBC::sqlQuery(con, SQLQuery)	A variável *SQLcommand* permite armazenar o resultado da consulta "*SQLQuery*" à ligação "*con*".
end.time	odbcClose(con) end.time <- Sys.time()	A variável *end.time* armazena o exato momento depois de terminada a ligação "con" à base de dados relacional.
SQL.time.taken	SQL.time.taken <- end.time - start.time	A variável *SQL.time.taken* indica a diferença de tempo entre as variáveis "*end.time*" e "*start.time*", permitindo assim obter o desempenho da consulta efetuada.

Quando se executa a conexão à base de dados relacional através da função "*odbcConnect*", o primeiro argumento refere-se à origem de dados. Esta origem de dados pode conter os valores "*Scn1*", "*Scn2*",

"*Scn3*" ou "*Scn4*", de acordo com o cenário a utilizar. O processo para a definição desta origem de dados pode ser encontrado em Anexo.

5.1.2 *Benchmarking* à Base de Dados em Grafo

A aplicação em *R* que permitiu medir o desempenho das consultas à base de dados em grafo foi desenvolvida utilizando a livraria "*RNeo4J*", que permite realizar todas as operações à base de dados em grafo *Neo4j* através da linguagem *R*. Utilizando a livraria "*RNeo4J*", foi possível construir a função "*Neo4JTest*" que permite realizar as consultas e medir o seu desempenho.

Nesta aplicação, existem algumas variáveis, presentes na tabela 30, que são fundamentais para a realização dos testes de desempenho:

Tabela 30 - Variáveis fundamentais na consulta à base de dados em grafo.

Variável	Exemplo da utilização	Descrição
start.time	start.time <- Sys.time()	A variável *start.time* permite armazenar o exato momento em que o teste começa a ser realizado.
CypherQuery	CypherQuery <- "MATCH (n:Flight) RETURN n"	A variável *CypherQuery* contém o *script* da consulta à base de dados em Grafo. São utilizadas nesta consulta as *queries* definidas no ponto 5.3.
graph	graph <- startGraph("http://localhost:7474/db/data/" , username = "neo4j", password = "qwerty")	A variável *graph* permite armazenar uma conexão com a base de dados em Grafo que é realizada através da função "*StartGraph*", onde no primeiro parâmetro é fornecida informação sobre o "*URL*" da ligação, o segundo parâmetro é o utilizador (por defeito o *username* é "neo4j") e o terceiro parâmetro é a palavra-passe do utilizador.
CypherCommand	CypherCommand <- cypher(graph, CypherQuery)	A variável *CypherCommand* permite armazenar o resultado da função *cypher*, isto é o resultado da ligação "*graph*" com a consulta "*CypherQuery*", que são os dois argumentos dessa função.

Variável	Exemplo da utilização	Descrição
end.time	end(graph) end.time <- Sys.time()	A variável *end.time* armazena o exato momento depois de terminada a ligação "graph" à base de dados relacional.
Cypher.time.taken	Cypher.time.taken <- end.time - start.time	A variável *Cypher.time.taken* indica a diferença de tempo entre as variáveis "*end.time*" e "*start.time*", permitindo assim obter o desempenho da consulta efetuada.

5.2 Cenários

Para a realização dos testes de desempenho às duas bases de dados (*SQLServer* e *Neo4J*) foram criados 4 cenários, ordenados de acordo com a quantidade de dados. Os cenários apenas variam na quantidade de dados, sendo que a essência dos dados é a mesma.

5.2.1 Cenário 1

O cenário 1 é o cenário que apresenta menor quantidade de anos, contendo apenas os voos referentes ao ano de 1999. Contém a seguinte quantidade de registos:

- 5 360 018 registos de voos do ano de 1999;
- 5 030 registos de aviões;
- 1 492 registos de companhias aéreas;
- 3 377 registos de aeroportos;
- Total de registos: 5 369 917 registos.

5.2.2 Cenário 2

O cenário 2 é o cenário que apresenta a terceira maior quantidade de anos, contendo os voos do ano 1995 a 1999. Contém a seguinte quantidade de registos:

- 26 317 534 registos de voos do ano de 1999 até 1999;
- 5 030 registos de aviões;
- 1 492 registos de companhias aéreas;
- 3 377 registos de aeroportos;

- Total de registos: 26 327 433 registos.

5.2.3 Cenário 3

O cenário 3 é o cenário que apresenta a segunda maior quantidade de anos, contendo os voos do ano 1995 a 2004. Contém a seguinte quantidade de registos:

- 56 083 667 registos de voos do ano de 1999 até 2004;
- 5 030 registos de aviões;
- 1 492 registos de companhias aéreas;
- 3 377 registos de aeroportos;
- Total de registos: 56 093 566 registos.

5.2.4 Cenário 4

O cenário 4 é o cenário que apresenta a maior quantidade de anos, contendo os voos do ano 1995 a 2008. Contém a seguinte quantidade de registos:

- 84 210 624 registos de voos do ano de 1999 até 2008;
- 5 030 registos de aviões;
- 1 492 registos de companhias aéreas;
- 3 377 registos de aeroportos;
- Total de registos: 84 220 523 registos.

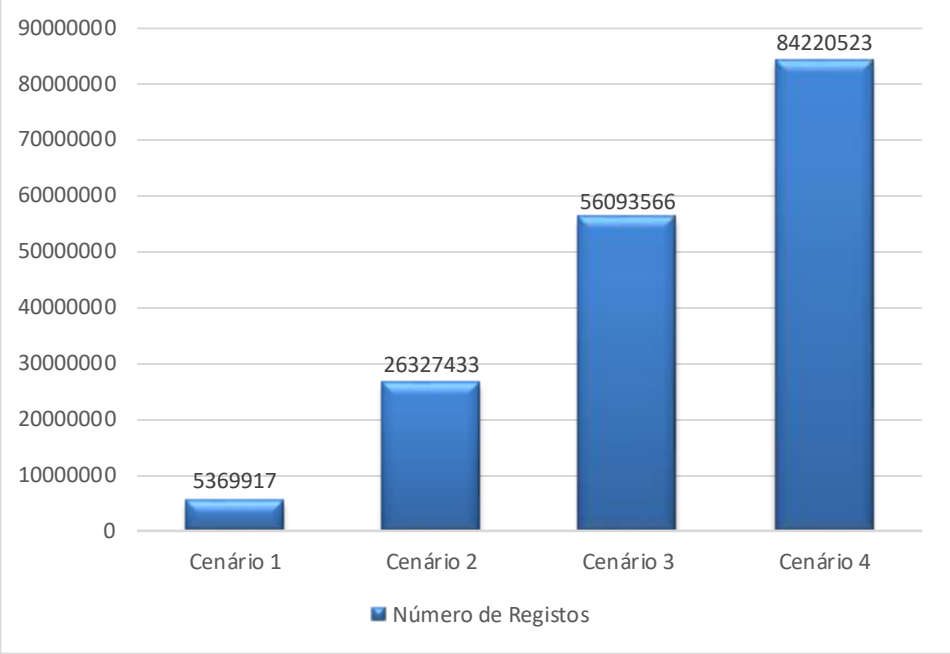

Figura 44 - Número de registos por cenário.

5.3 Consultas às Bases de Dados

Para submeter a testes os 4 cenários anteriormente descritos, foram criadas 4 consultas diferentes, que têm como objetivo verificar o seu comportamento perante diferentes quantidades de dados (Cenários). As consultas à base de dados relacional encontram-se na linguagem *SQL*, que é a linguagem *standard* de entre as bases de dados relacionais, enquanto que as consultas à base de dados em grafo se encontram em *Cypher*. Segundo testes realizados pelos autores (Angles, 2016) e (Holzschuher & Peinl, 2016), estes demonstram que a linguagem *Cypher* é a que tem melhor desempenho e a mais simples e intuitiva em termos de sintaxe e semântica. A linguagem *Gremlin*, a segunda linguagem mais conhecida de entre as bases de dados em grafo, desempenha bastante bem em *queries* simples, sendo considerada uma linguagem de baixo nível (Holzschuher & Peinl, 2016). Em várias comparações de desempenho, esta tem desempenhado pior que a principal linguagem de consulta utilizada pelo *Neo4J*, o *Cypher* (Holzschuher & Peinl, 2013) (Wood, 2012) (de Oliveira Santos, et al., 2016).

As consultas devolvem exatamente o mesmo resultado, apesar de estarem em diferentes linguagens de consulta. O formato do resultado de todas as consultas na aplicação de medição de desempenho construída é "*data.table*".

5.3.1 Consulta 1

Esta é uma das mais simples consultas que se pode realizar à base de dados. O seu objetivo é "Selecionar a matrícula dos aviões, assim como a média do tempo de atraso de um avião em todos os seus voos, de modo a verificar quais têm a tendência em se atrasar mais à chegada". Esta consulta é representada em *SQL* pelo código da figura 45.

```
SELECT AVG(arrDelay), tailNum
FROM Flight
GROUP BY tailNum
```

Figura 45 - Script da consulta 1 em SQL.

Em *SQL*, esta consulta utiliza uma função, o "*AVG*" que determina a média de um conjunto de valores. É ainda utilizado um "*Group By*", que permite agrupar todos os valores com o mesmo *tailNum* e calcular, assim, a média.

Em *Cypher*, esta consulta é representada pelo código da figura 46.

```
MATCH (n:Flight)
RETURN DISTINCT n.tailNum, avg(toInt(n.arrDelay))
```

Figura 46 - Script da consulta 1 em Cypher.

Em *Cypher*, esta consulta utiliza duas funções pré-definidas, o "*AVG*", que permite fazer a média de cada *tailNum* distinto e o "*ToInt*", que permite ler e calcular a propriedade *arrDelay* como um número inteiro. Estas duas consultas utilizam funções pré-definidas das suas linguagens, sendo esse o principal objetivo, ou seja, verificar qual o desempenho dos dois sistemas de base de dados perante estas funções predefinidas em cada um dos 4 cenários em estudo.

5.3.2 Consulta 2

Esta é uma *querie* já com alguma complexidade, uma vez que contém buscas transversais a quatro tabelas (*Airplane*, *Manufacturer*, *Airline* e *Airport*), sendo o seu objetivo "Selecionar todos os aviões, juntamente com o seu fabricante, companhia aérea e a distância percorrida para voos realizados no ano de 1999 cujo aeroporto de destino é "*Los Angeles International*"".

Esta consulta é representada em *SQL* pelo código presente na figura 47.

```
SELECT A.tailNum, M.manufacturer, L.[description], F.distance
FROM Flight F
INNER JOIN Airplane A
ON F.tailNum = A.tailNum
INNER JOIN Manufacturer M
ON M.idManufacturer=A.idManufacturer
INNER JOIN Airline L
ON F.uniqueCarrier=l.idAirline
INNER JOIN Airport P
ON P.iata=F.dest
WHERE P.airport='"Los Angeles International"' and F.[year]=1999
```

Figura 47 - Script da consulta 2 em SQL.

Esta consulta *SQL* é algo complexa, uma vez que faz buscas transversais a 5 tabelas, e ainda tem duas restrições através da condição "*Where*".

A consulta 2 é representada em *Cypher* pelo código presente na figura 48.

```
MATCH (L:Airlines)-[:By]-(F:Flight)-[:To]-(A:Airport {airport: "Los Angeles International"}) WITH L,F,A
MATCH (F:Flight {year: "1999"})-[:On]-(T:Airplane) with F, L, T
RETURN F.tailNum, T.manufacturer, L.description, F.distance
```

Figura 48 - Script da consulta 2 em Cypher.

Esta *querie* em *Cypher* é considerada complexa, uma vez que são realizados dois "*Match*", juntamente com dois "*With*", que permite que as consultas sejam encadeadas uma com a outra.

Estas duas consultas têm como objetivo verificar o desempenho em ambas as bases de dados perante consultas a várias tabelas ou tipos de nodos.

5.3.3 Consulta 3

Esta consulta é mais complexa que a consulta 2, uma vez que necessita de aceder a todas as tabelas e retornar resultados de todas essas tabelas. Com esta consulta pretende-se "Selecionar todos os voos com as várias informações destes, como a sua identificação, tipo de voo, o modelo do avião, e o número de voo) da companhia aérea "*Delta Air Lines Inc.*" que tiveram origem no aeroporto com o código "*iata*" "*ATL*"."

Esta consulta é representada em *SQL* pelo código presente na figura 49.

```
SELECT E.[type], A.tailNum, At.[type], Mo.model, M.Manufacturer, P.airport,
F.flightNum, L.[description]
FROM Flight F
INNER JOIN Airplane A
ON F.tailNum = A.tailNum
INNER JOIN Airline L
ON L.idAirline = F.uniqueCarrier
INNER JOIN Airport P
ON P.iata = F.origin
INNER JOIN Plane_Type Pl
ON Pl.idPlaneType = A.idPlaneType
INNER JOIN Model Mo
ON Mo.idModel = A.idModel
INNER JOIN Manufacturer M
ON M.idManufacturer = A.idManufacturer
INNER JOIN Engine_type E
ON E.idEngineType = A.idEngineType
INNER JOIN Aircraft_type At
ON At.idAircraftType = A.idAircraftType
WHERE P.iata = 'ATL' AND  L.[description] = '"Delta Air Lines Inc."'
```

Figura 49 - Script da consulta 3 em SQL.

Como podemos visualizar na figura 50, é realizado uma consulta transversal a todas as tabelas da base de dados relacional, tornando esta consulta muito pesada e complexa.

```
MATCH (a:Airline {description:"Delta Air Lines Inc."})-[:By]-(f1:Flight)-
[:From]-(p:Airport {iata:"ATL"}),(f1:Flight)-[:On]-(ap:Airplane)
RETURN ap.engine_type, ap.tailnum, ap.aircraft_type, ap.model,
ap.type, ap.manufacturer, p.airport, f1.idFlight, a.description
```

Figura 50 - Script da consulta 3 em Cypher.

Esta consulta em *Cypher* é considerada complexa, uma vez que são realizados dois "*Match*", sendo que um deles consulta 3 nodos e duas relações diferentes, ao passo que o outro "*Match"* realiza consultas em dois nodos. Este segundo "*Match* (f1:Flight)" destaca-se por utilizar o "*f1*", ou seja, são considerados todos os voos que estão presentes no "*Match"* anterior.

O principal objetivo da consulta 3 era verificar o comportamento da base de dados em grafos a uma consulta que na base de dados relacional é considerada complexa, pois tem consultas a todas as tabelas existentes no modelo.

5.3.4 Consulta 4

Esta consulta tem como objetivo retornar não uma grande quantidade de dados, mas sim uma grande diversidade de atributos. Com esta consulta pretende-se "Selecionar toda a informação disponível relativamente aos voos, companhia aérea e dos aviões N382UA e N381UA da "*United Air Lines Inc.*", onde o tempo de voo foi menor que 200 minutos." Esta consulta é representada em *SQL* pelo código presente na figura 51.

```
SELECT *
FROM Flight F
INNER JOIN Airline L
ON L.idAirline = F.uniqueCarrier
INNER JOIN Airplane A
ON A.tailNum = F.tailNum
WHERE L.[description]='"United Air Lines Inc."' AND
(A.tailNum='N382UA' OR A.tailNum='N381UA')
AND  F.airTime<200
```

Figura 51 - Script da consulta 4 em SQL.

Em *SQL* são consultadas três tabelas, e são devolvidos todos os resultados que vão de encontro às condições estabelecidas.

```
MATCH (L:Airline {description:"United Air Lines Inc."})-[:By]-(F:Flight)-[:On]-(A:Airplane
{tailnum:"N381UA"})
WHERE F.airTime<200
RETURN F, L, A
UNION
MATCH (L:Airline {description:"United Air Lines Inc."})-[:By]-(F:Flight)-[:On]-(A:Airplane
{tailnum:"N382UA"})
WHERE F.airTime<200
RETURN F.idFlight, F.year, F.month, F.dayOfMonth, F.dayOfWeek, F.depTime, F.crsDepTime,
F.arrTime, F.crsArrTime, F.flightNum, F.actualElapsedTime, F.crsElapsedTime, F.airTime,
F.arrDelay, F.depDelay, F.distance, F.taxiIn, F.taxiOut, D.dest, F.origin, F.tailNum, A.uniqueCarrier,
A.idAirline, A.description, L.tailNum, L.issueDate, L.status, L.year, L.idPlaneType, L.idManufacturer,
L.idModel, L.idAircraftType, L.idEngineType
```

Figura 52 - Script da consulta 4 em Cypher.

Esta consulta (figura 52) tem algum nível de complexidade, pois além de dois "*Match*" já com algumas restrições, faz ainda uma união dos dois resultados.

O objetivo da consulta 4 é verificar o comportamento dos sistemas de bases de dados perante o facto de devolver uma grande quantidade de atributos diferentes.

5.4 Resultados do *Benchmarking*

Todos os resultados do *Benchmarking* são apresentados neste capítulo. De notar que todos esses resultados se encontram registados de acordo com o Sistema Internacional de unidades (SI), isto é, estão registados em segundos (s).

5.4.1 Consulta 1

Tabela 31 - Resultados Consulta 1 - Cenário 1.

Cenário 1	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo Médio de Consulta (s)
1 utilizador	0,327	0,327	4,073	4,073
10 utilizadores	3,438	0,343	43,871	4,387
25 utilizadores	9,002	0,360	120,326	4,813
50 utilizadores	20,209	0,404	247,384	4,947
100 utilizadores	43,058	0,430	510,040	5,100
Tempo médio por consulta	0,3728		4,664	

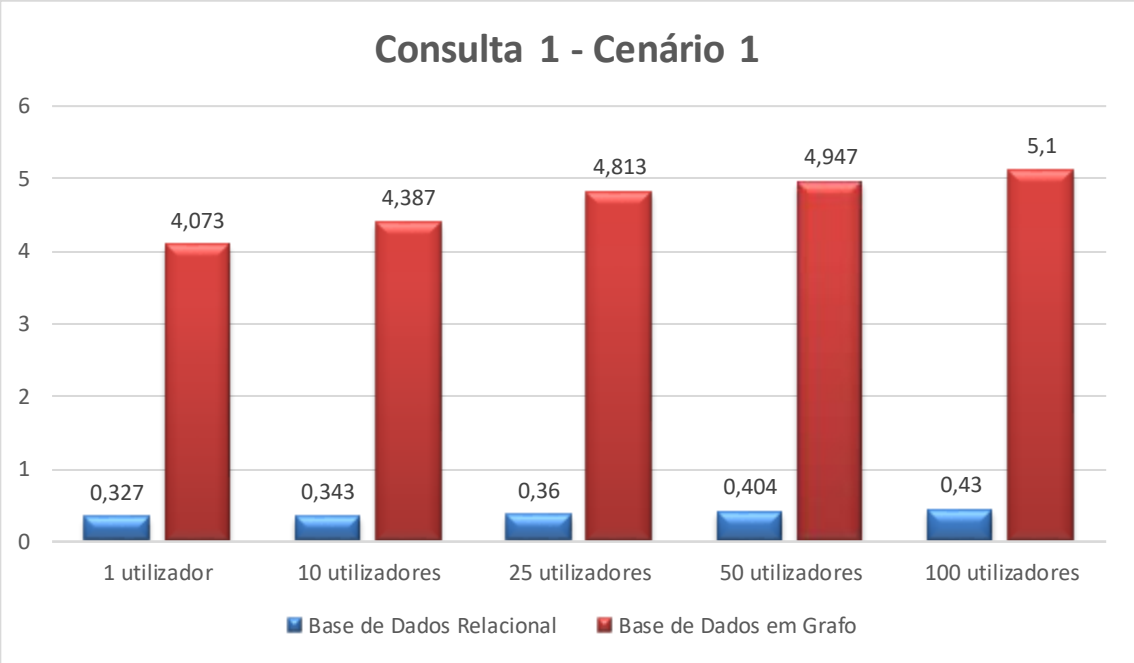

Figura 53 - Gráfico dos resultados da Consulta 1 - Cenário 1.

Tabela 32 - Resultados Consulta 1 - Cenário 2.

Cenário 2	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	1,284	1,284	20,010	20,010
10 utilizadores	13,095	1,309	204,203	20,420
25 utilizadores	35,112	1,404	535,696	21,427
50 utilizadores	74,729	1,494	1109,586	22,191
100 utilizadores	152,762	1,527	2330,472	23,304
Tempo médio por consulta	1,404		21,470	

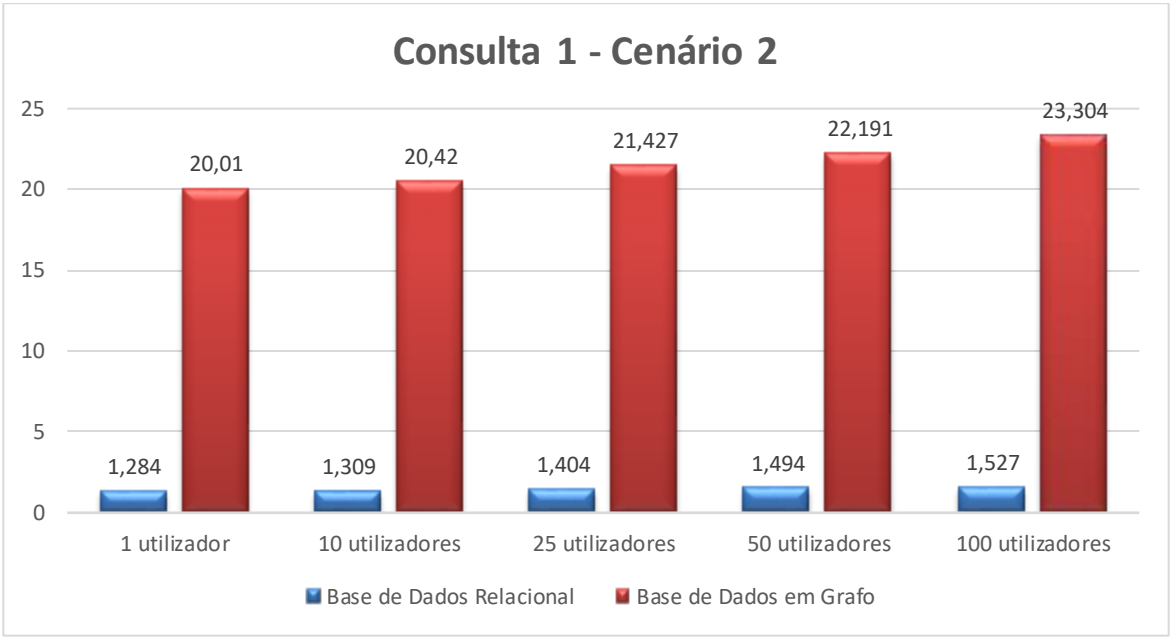

Figura 54 - Gráfico dos resultados da Consulta 1 - Cenário 2.

Tabela 33 - Resultados Consulta 1 - Cenário 3.

Cenário 3	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	4,239	4,239	50,550	50,550
10 utilizadores	43,612	4,361	519,544	51,954
25 utilizadores	110,949	4,437	1320,364	52,814
50 utilizadores	240,739	4,814	2698,933	53,978
100 utilizadores	496,582	4,965	5851,689	58,516
Tempo médio por Consulta	4,5632		53,5624	

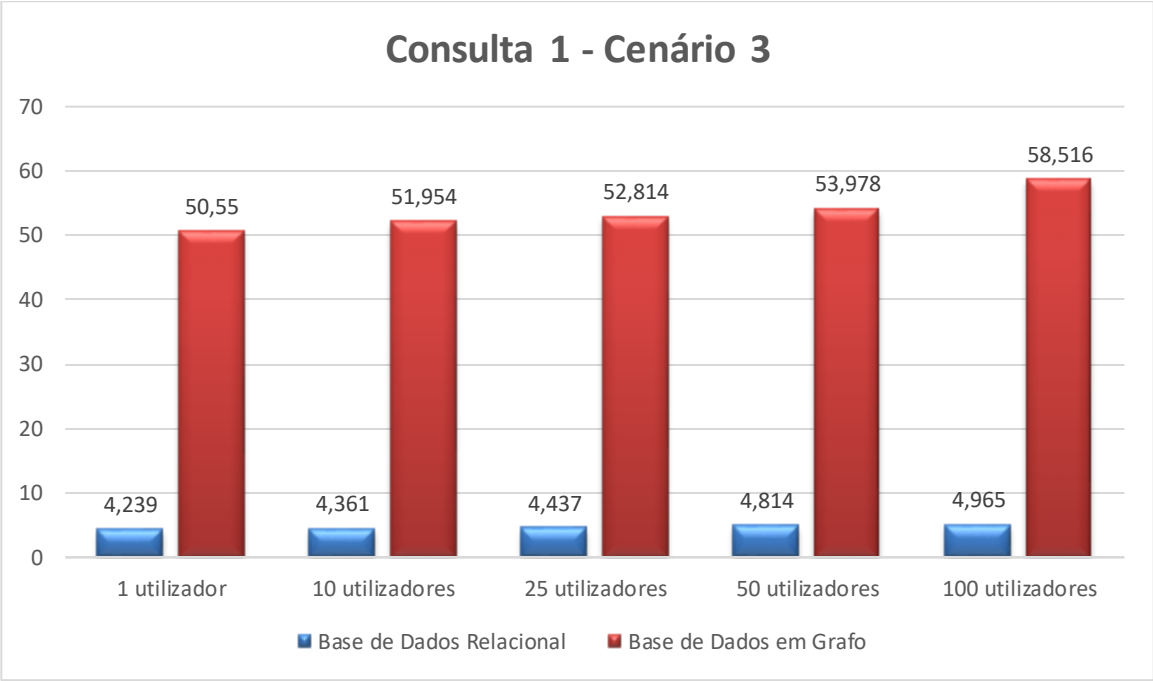

Figura 55 - Gráfico dos resultados da Consulta 1 - Cenário 3.

Tabela 34 - Resultados Consulta 1 - Cenário 4.

Cenário 4	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	7,274	7,274	265,392	265,392
10 utilizadores	73,873	7,387	2735,065	273,506
25 utilizadores	193.412	7,736	7224,85	288.994
50 utilizadores	389,801	7,796	14549,17	290,983
100 utilizadores	843,301	8,433	30158,72	301,587
Tempo médio por consulta	7,7252		284,0924	

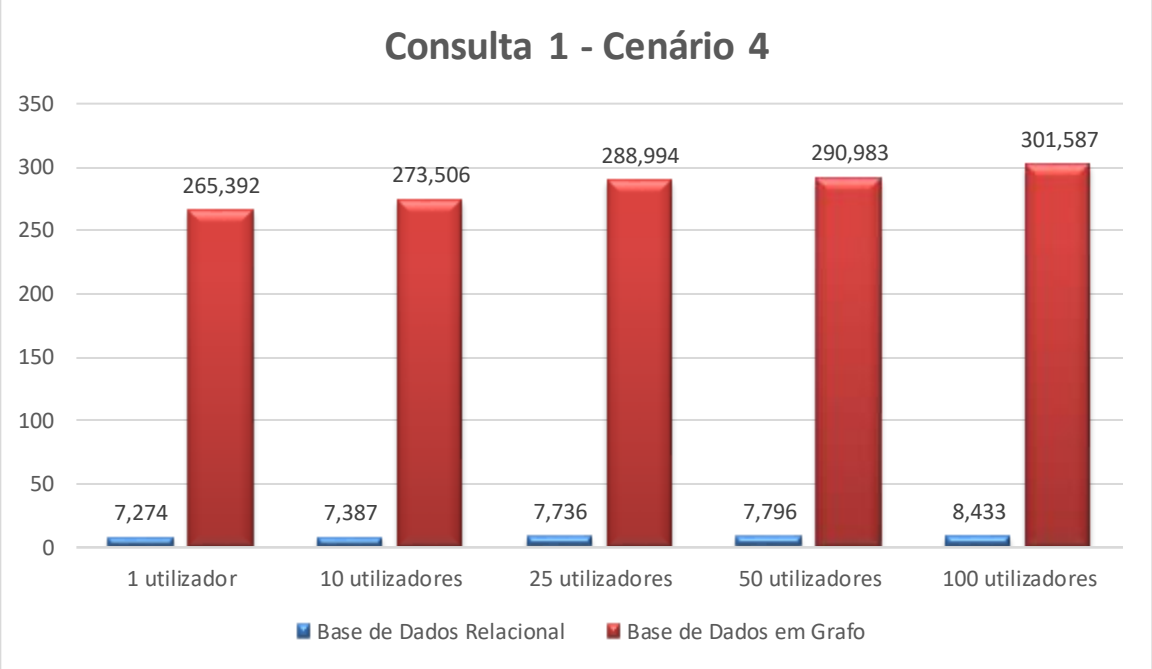

Figura 56 - Gráfico dos resultados da Consulta 1 - Cenário 4.

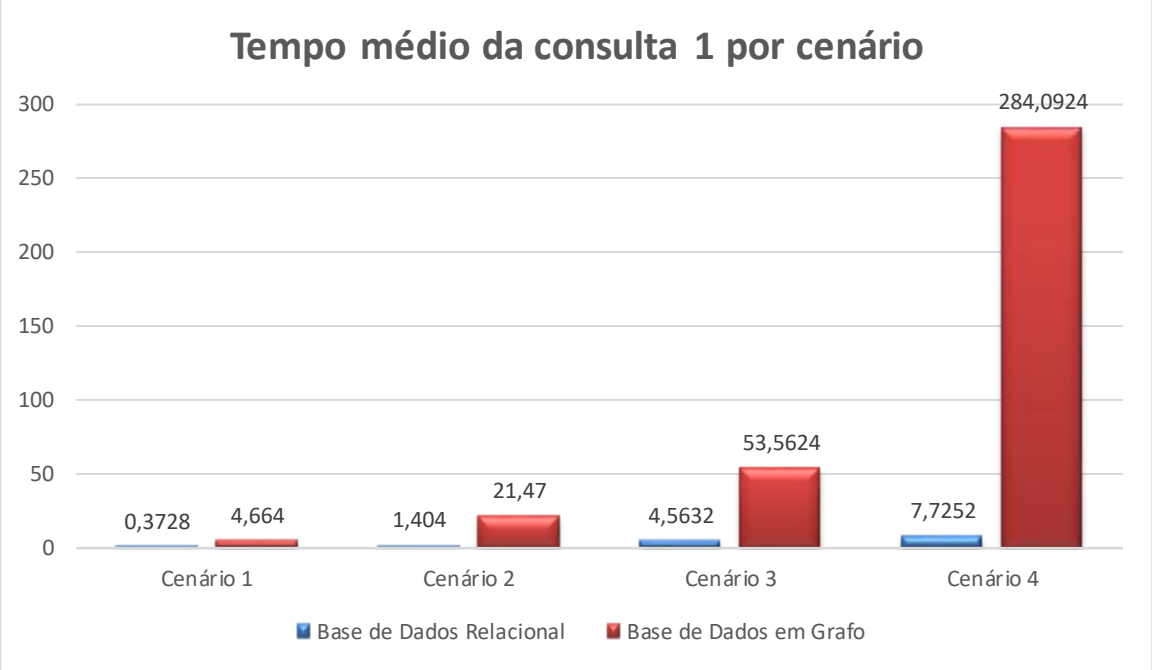

Figura 57 - Tempo médio da consulta 1 por cenário.

Na análise efetuada à consulta 1, pode verificar-se que a base de dados relacional tem um desempenho bastante melhor que a base de dados em grafos. Estes resultados são perfeitamente justificáveis, o que demonstra que a base de dados em grafo é desenhada para desempenhar bem com dados interligados e não com funções predefinidas. Repare-se também o aumento médio do tempo das consultas em ambas as bases de dados, sendo que este aumento é exponencial ao longo do aumento do tamanho dos cenários. Quanto mais dados, maior será o tempo de consulta, o que demonstra que este tipo de consultas não é ideal para ser realizado na base de dados em grafo, sobretudo na presença de muitos dados. Em todos os cenários, importa realçar que há também um pequeno aumento no tempo de consulta sempre que o número de utilizadores aumenta. Conclui-se que para consultas que envolvam utilizar funções pré-definidas das bases de dados da qual a média é um exemplo (*AVG*), a relacional desempenha bastante melhor que o modelo em grafo.

5.4.2 Consulta 2

Tabela 35 - Resultados Consulta 2 - Cenário 1.

Cenário 1	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	2,737	2,737	3,090	3,090
10 utilizadores	27,860	2,786	30,881	3,0881
25 utilizadores	70,731	3,229	82,664	3,306
50 utilizadores	178,369	3,567	173,156	3,463
100 utilizadores	399,027	3,990	350,097	3,500
Tempo médio por consulta	3,2618		3,2894	

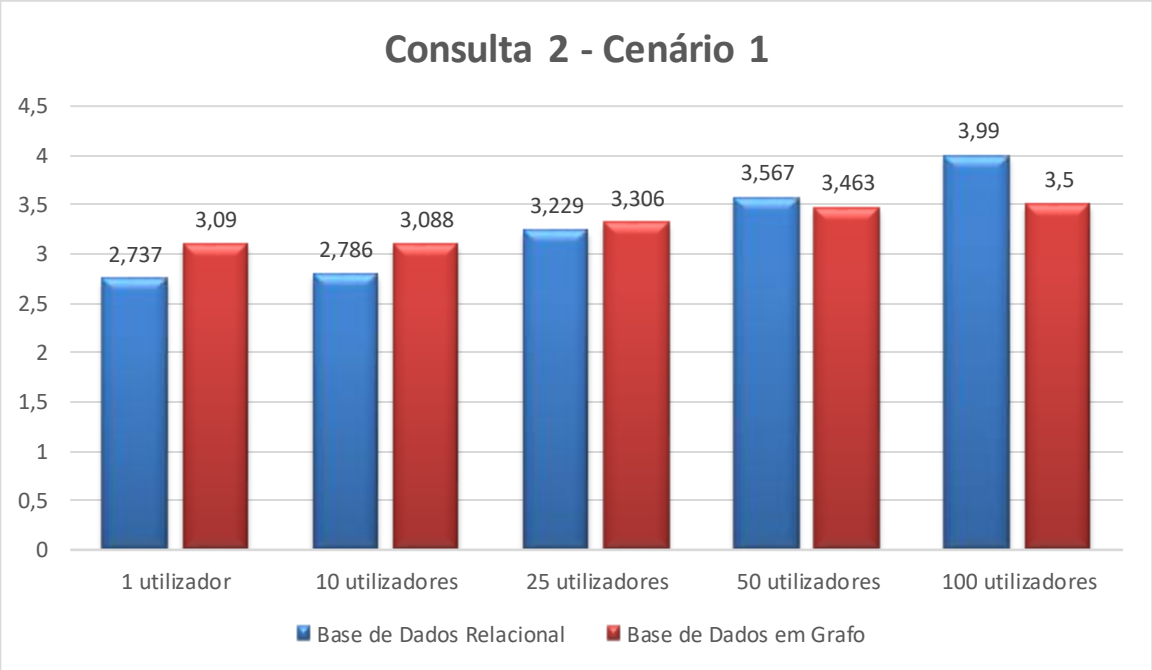

Figura 58 - Gráfico dos resultados da Consulta 2 - Cenário 1.

Tabela 36 - Resultados Consulta 2 - Cenário 2.

Cenário 2	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	3,352	3,352	3,409	3,409
10 utilizadores	35,976	3,597	36,670	3,667
25 utilizadores	91,733	3,669	94,022	3,760
50 utilizadores	201,002	4,020	188,578	3,771
100 utilizadores	498,649	4,986	389,948	3,809
Tempo médio por consulta	3,9248		3,6832	

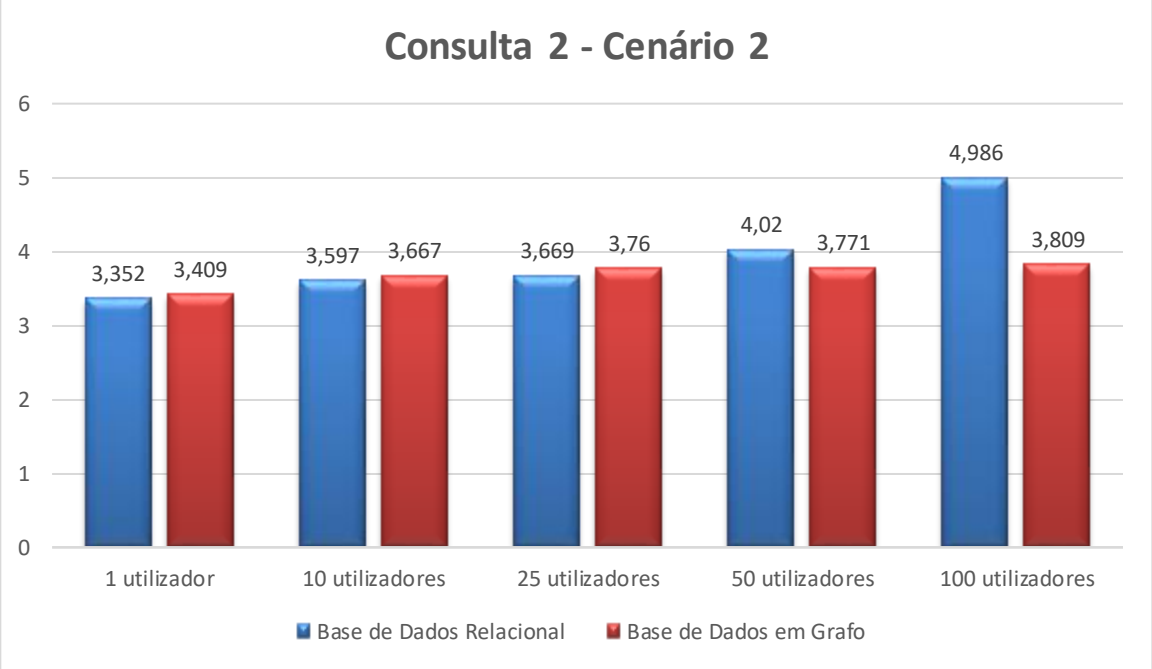

Figura 59 - Gráfico dos resultados da Consulta 2 - Cenário 2.

Tabela 37 - Resultados Consulta 2 - Cenário 3.

Cenário 3	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	5,010	5,010	5,065	5,065
10 utilizadores	53,164	5,316	51,3060	5,130
25 utilizadores	136,652	5,466	131,920	5,276
50 utilizadores	275,126	5,502	268,245	5,364
100 utilizadores	567,109	5,671	549,380	5,493
Tempo médio por consulta	5,393		5,2656	

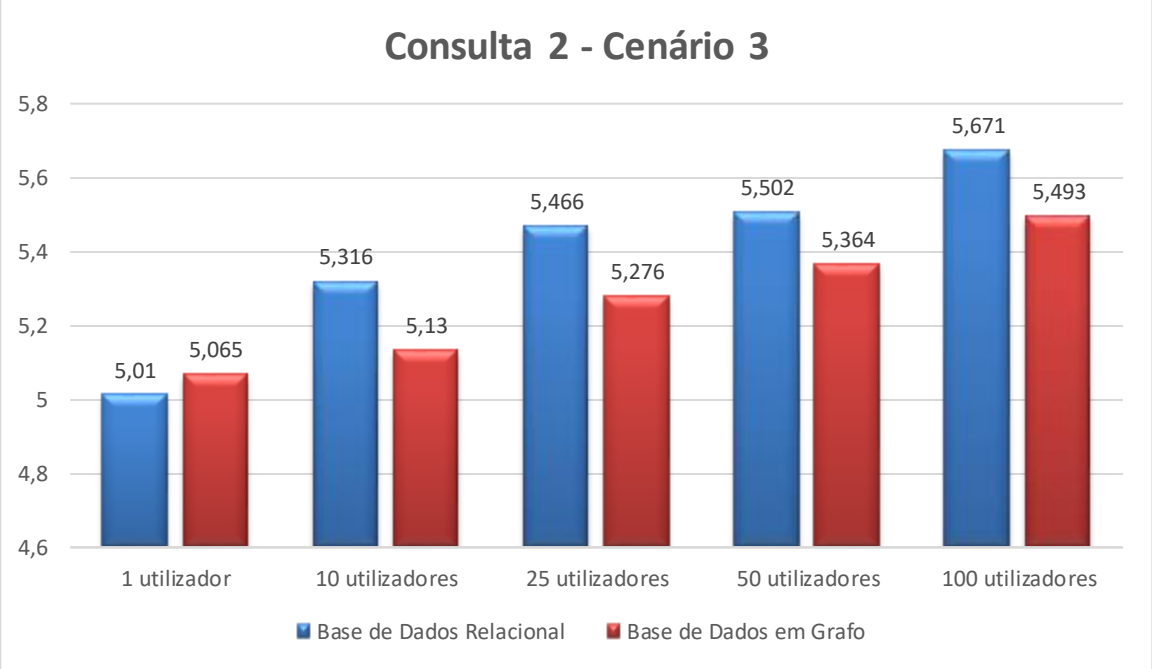

Figura 60 - Gráfico dos resultados da Consulta 2 - Cenário 3.

Tabela 38 - Resultados Consulta 2 - Cenário 4.

Cenário 4	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	6,798	6,798	6,765	6,765
10 utilizadores	66,618	6,761	67,890	6,789
25 utilizadores	173,673	6,946	172,33	6,893
50 utilizadores	353,765	7,075	352,316	7,046
100 utilizadores	742,789	7,427	732,699	7,326
Tempo médio por consulta	7,0014		6,9638	

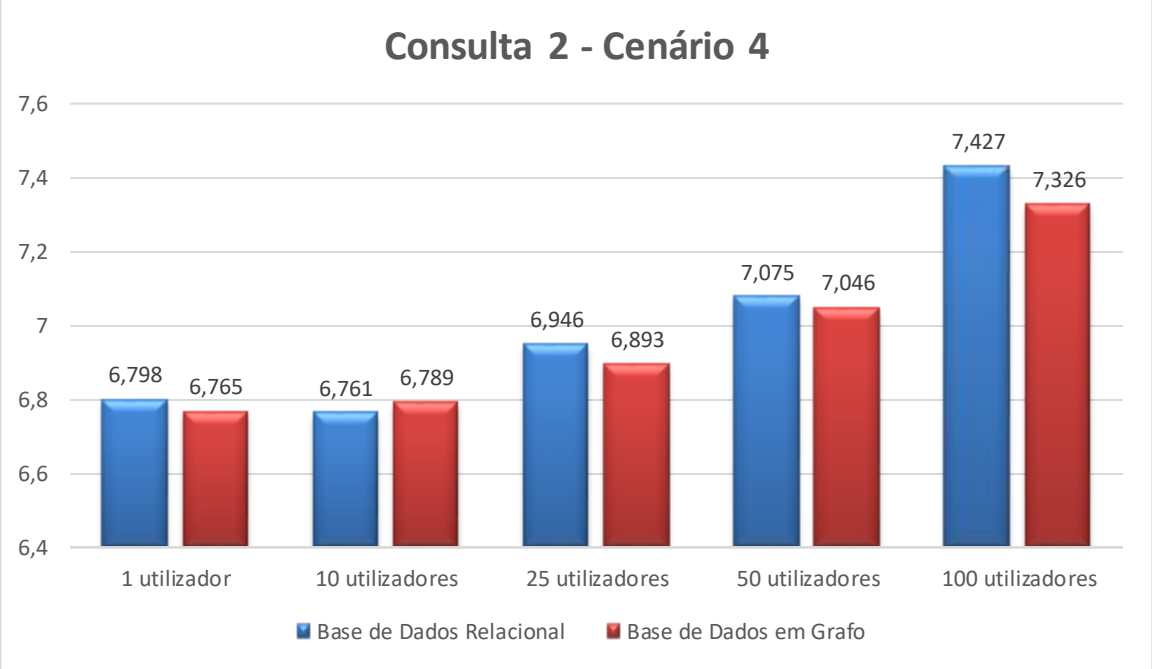

Figura 61 - Gráfico dos resultados da Consulta 2 - Cenário 4.

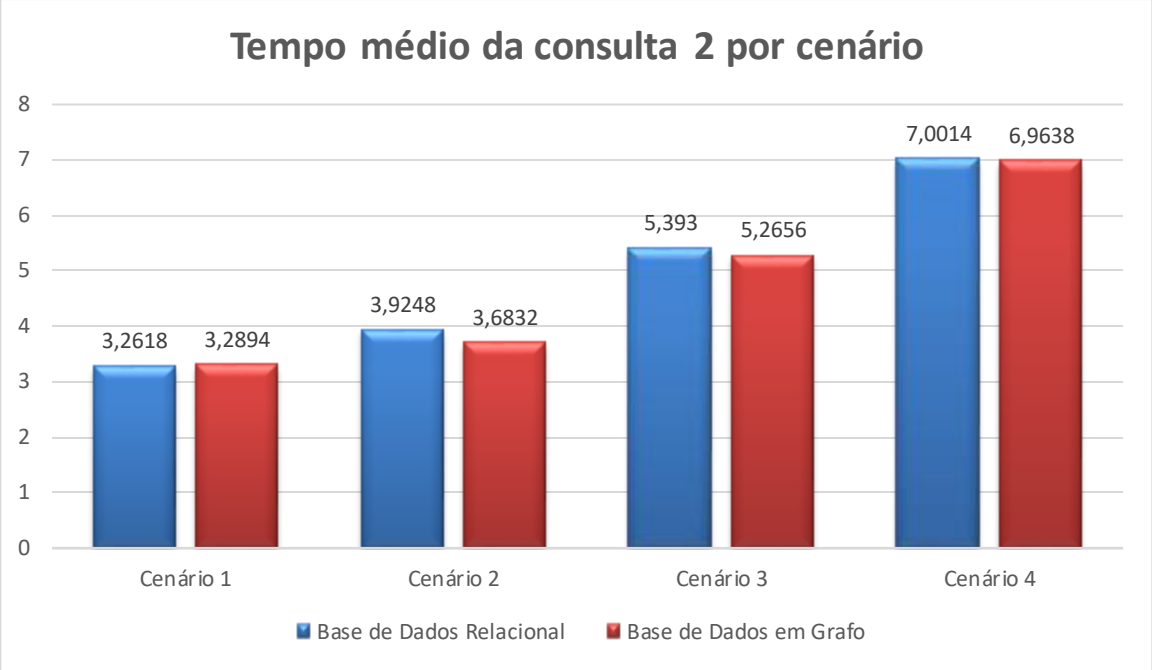

Figura 62 - Tempo médio da consulta 2 por cenário.

Na análise efetuada à consulta 2, pode verificar-se um resultado muito curioso. Ambas as consultas obtêm resultados muito semelhantes. Os resultados foram os seguintes:

- Cenário 1 - As consultas à base de dados relacional obtiveram melhor resultado quando o número de utilizadores era 1, 10 e 25 utilizadores. Para 50 e 100 utilizadores, a base de dados em grafo obteve melhor desempenho, de onde se conclui que para muitos utilizadores e não muitos dados, a base de dados em grafo desempenha melhor. Se houver poucos utilizadores, a base de dados relacional desempenha melhor. No que toca ao tempo médio de todas as consultas, a base de dados relacional desempenhou melhor.

- Cenário 2 - As consultas à base de dados relacional obtiveram melhor resultado quando o número de utilizadores era 1, 10 e 25 utilizadores. Para 50 e 100 utilizadores, a base de dados em grafo obteve melhor desempenho, de onde se conclui que, para muitos utilizadores e não muitos dados, a base de dados em grafo tem um melhor desempenho. De notar que o tempo de consulta à base de dados em grafo manteve-se quase constante independentemente do número de utilizadores, ao contrário das consultas à base de dados relacional, que foi crescendo exponencialmente, ainda que num grau

baixo. De uma maneira geral, o tempo de consulta médio independentemente do número de utilizadores foi melhor na base de dados em grafo.

- Cenário 3 - As consultas à base de dados relacional apenas tiveram melhor desempenho quando o número de utilizadores era 1. Todas as consultas com mais que um utilizador, obtiveram melhores resultados quando consultaram a base de dados em grafo.

- Cenário 4 - Apenas na situação de uma consulta com 10 utilizadores a base de dados relacional desempenhou melhor. Em todas as outras situações a base de dados em grafo desempenhou melhor. De reparar que neste cenário existe a tendência para um crescimento exponencial de maior grau para as bases de dados relacionais, concluindo que quanto mais utilizadores, melhor desempenha a base de dados em grafo.

No que toca ao tempo médio, as consultas à base de dados em grafo tiveram melhores desempenhos nos cenários 2, 3 e 4, concluindo que, de uma maneira geral, independentemente do número de utilizadores, quanto mais informação, melhor a base de dados desempenha.
Conclui-se então, que em consultas que envolvam "*joins*" na base de dados relacional, enquanto que na base de dados em grafo são representadas através de relações, estas têm melhor desempenho na base de dados em grafo, sobretudo se a quantidade de dados for muito elevada.

5.4.3 Consulta 3

Tabela 39 - Resultados Consulta 3 - Cenário 1.

Cenário 1	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	14,084	14,084	14,033	14,033
10 utilizadores	149,751	14,975	145,148	14,514
25 utilizadores	396,607	15,864	375,280	15,011
50 utilizadores	861,02	17,220	791,495	15,829
100 utilizadores	1791,61	17,916	1642,100	16,421
Tempo médio por consulta	16,0118		15,1616	

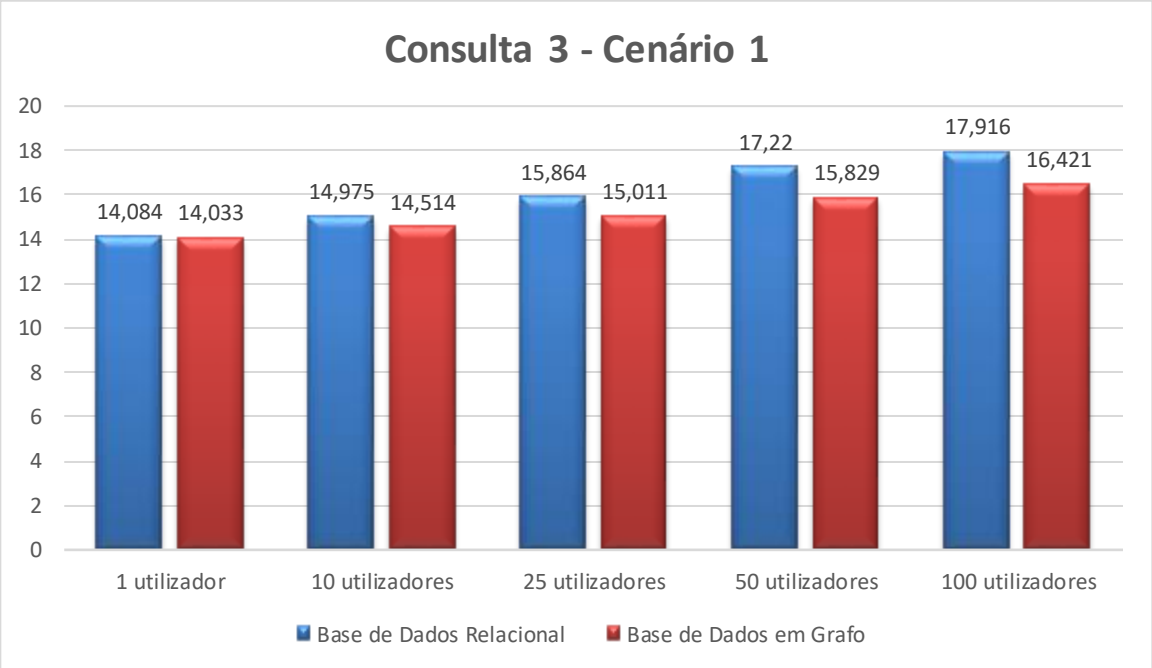

Figura 63 - Gráfico dos resultados da Consulta 3 - Cenário 1.

Tabela 40 - Resultados Consulta 3 - Cenário 2.

Cenário 2	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	55,151	55,151	50,300	50,300
10 utilizadores	569,124	56,912	513,223	51,322
25 utilizadores	1453,980	58,159	1333,010	53,320
50 utilizadores	3006,290	60,125	2802,255	56,045
100 utilizadores	6549,256	65,492	5808,060	58,080
Tempo médio por consulta	59,1678		53,8134	

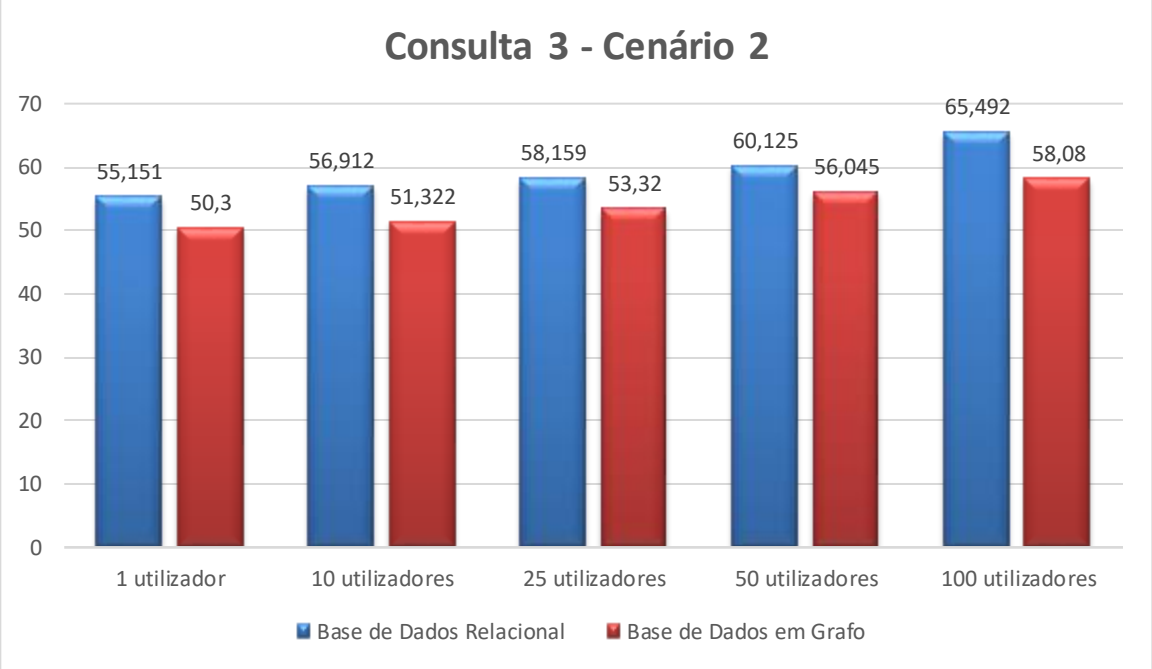

Figura 64 - Gráfico dos resultados da Consulta 3 - Cenário 2.

Tabela 41 - Resultados Consulta 3 - Cenário 3.

Cenário 3	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	120,605	120,605	121,680	114,680
10 utilizadores	1226,325	122,632	1176,057	117,605
25 utilizadores	3193,456	127,738	2978,812	119,152
50 utilizadores	6539,335	130,786	6162,405	123,248
100 utilizadores	13369,870	133,698	12620,980	126,209
Tempo médio por consulta	127,0918		120,1788	

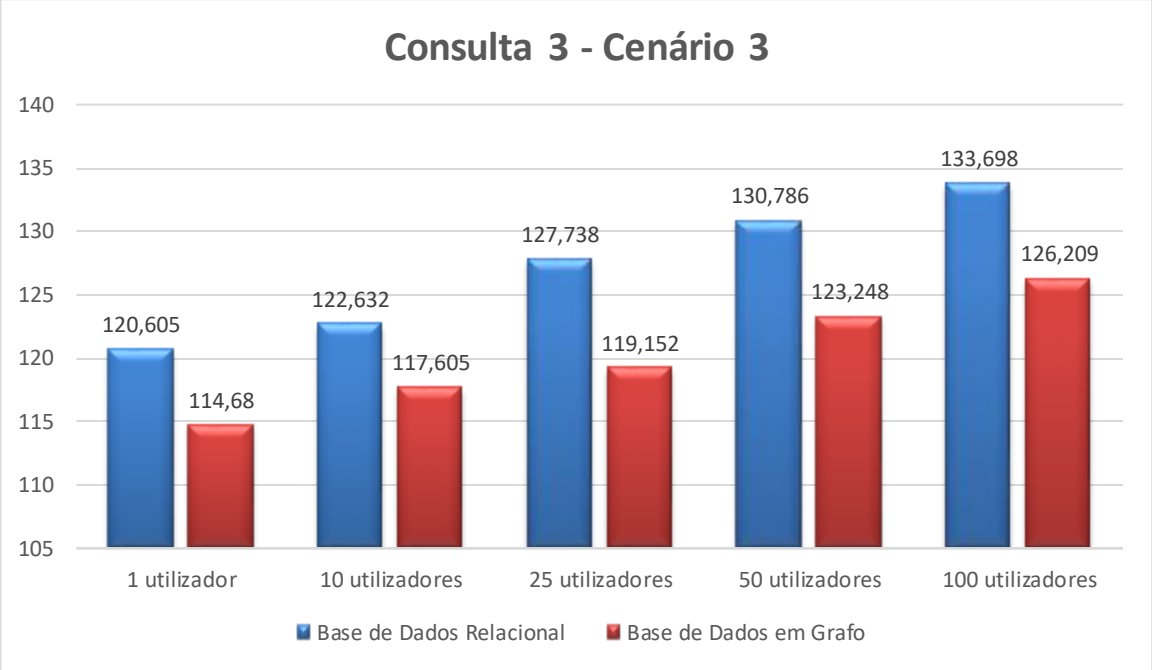

Figura 65 - Gráfico dos resultados da Consulta 3 - Cenário 3.

Tabela 42 - Resultados Consulta 3 - Cenário 4.

Cenário 4	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	172,125	172,125	154,131	154,131
10 utilizadores	1743,511	174,351	1570,917	157,091
25 utilizadores	4452,520	178,100	3982,292	159,291
50 utilizadores	9249,600	184,992	8139,010	162,780
100 utilizadores	19020,230	190,202	16523,830	165,238
Tempo médio por consulta	179,954		159,7062	

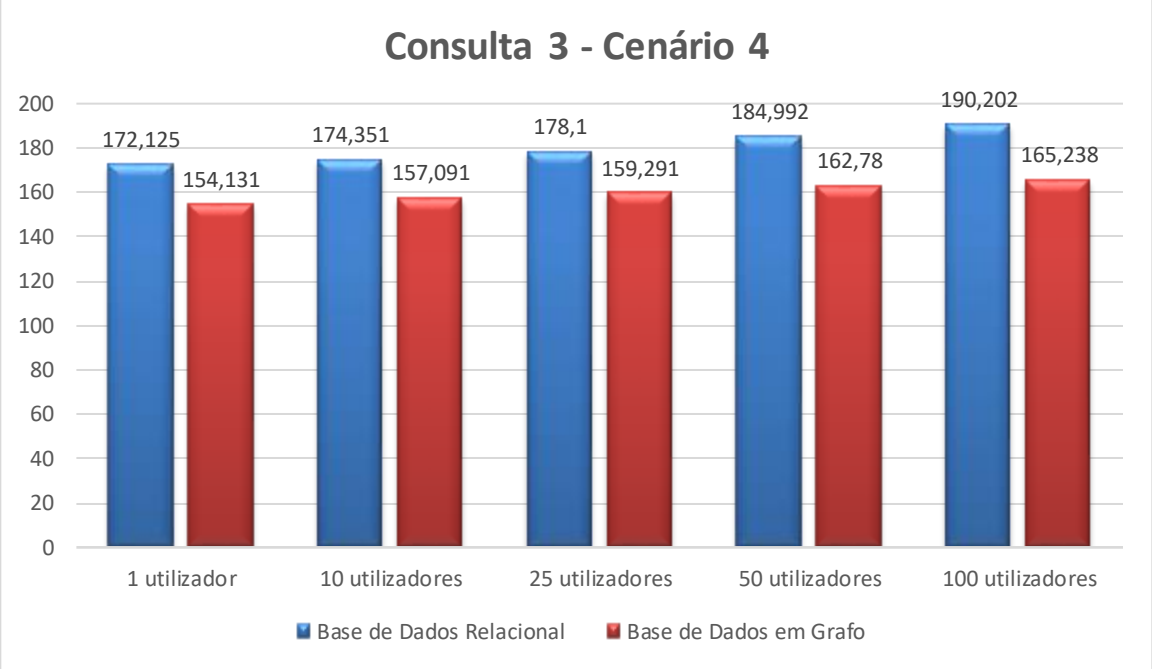

Figura 66 - Gráfico dos resultados da Consulta 3 - Cenário 4.

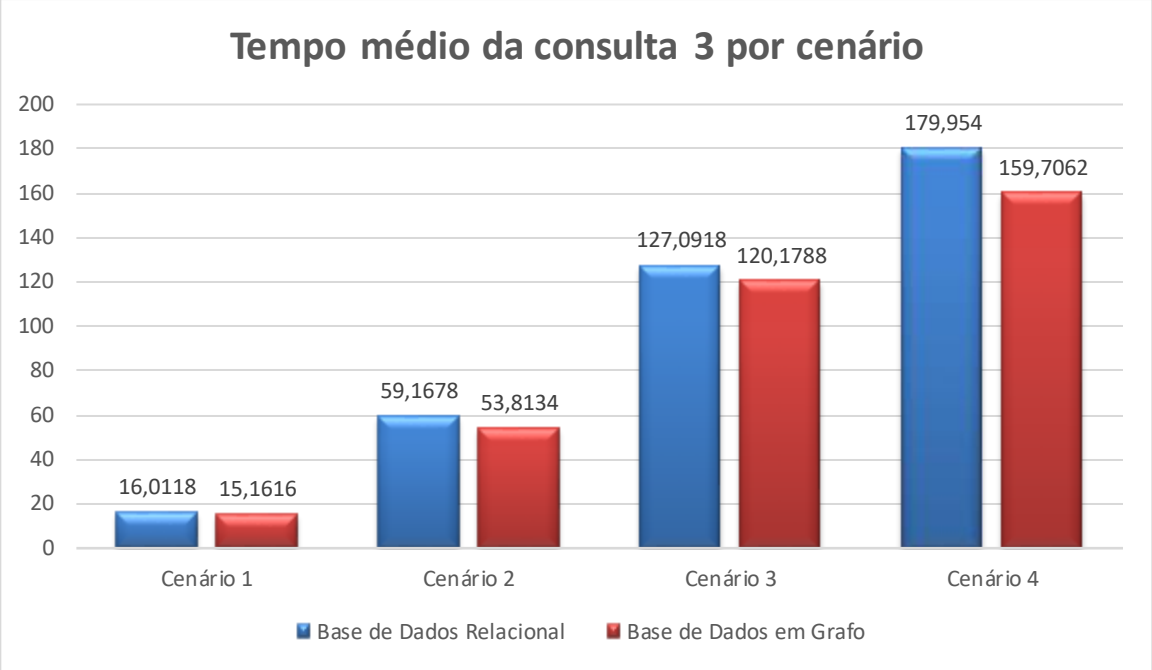

Figura 67 - Tempo médio da consulta 3 por cenário.

Na análise efetuada à consulta 3, pode verificar-se que a base de dados em grafos desempenhou sempre melhor do que a base de dados relacional em todos os cenários e com qualquer número de utilizadores. O tempo médio por consulta em cada cenário aumentou exponencialmente em ambos os sistemas de base de dados, contudo o grau de crescimento é maior na base de dados relacional, o que demonstra que a base de dados em grafo desempenha melhor perante grandes quantidades de dados que a base de dados relacional. Esta consulta acaba por comprovar o que foi comprovado na consulta 3, ou seja, que a base de dados em grafo desempenha melhor que a base de dados relacional quando se está perante muitos dados, sendo estes relacionáveis entre si, e que envolvam muitos "*Joins*" na consulta à base de dados relacional.

5.4.4 Consulta 4

Tabela 43 - Resultados Consulta 4 - Cenário 1.

Cenário 1	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	1,266	1,266	0,928	0,928
10 utilizadores	12,744	1,274	9,462	0,946
25 utilizadores	32,521	1,300	24,003	0,960
50 utilizadores	66,303	1,326	48,196	0,963
100 utilizadores	134,3	1,343	97,499	0,974
Tempo médio por consulta	1,3018		0,9542	

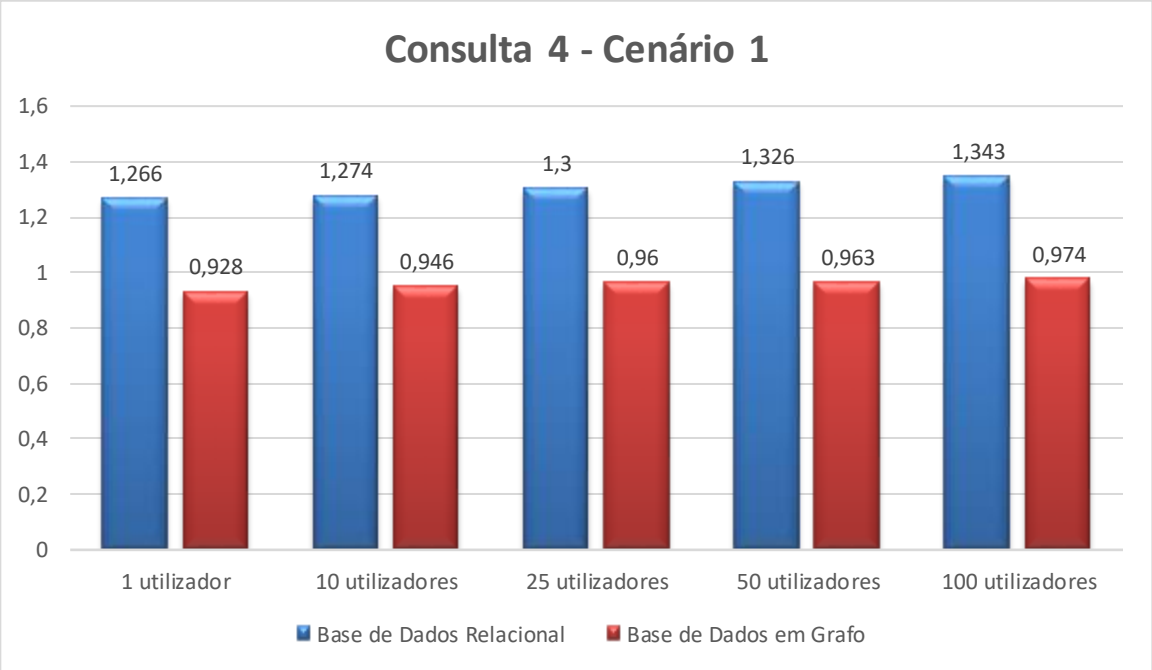

Figura 68 - Gráfico dos resultados da Consulta 4 - Cenário 1.

Tabela 44 - Resultados Consulta 4 - Cenário 2.

Cenário 2	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	5,267	5,267	3,009	3,009
10 utilizadores	53,944	5,394	32,606	3,260
25 utilizadores	135,492	5,419	91,039	3,641
50 utilizadores	281,921	5,638	192,632	3,852
100 utilizadores	586,854	5,868	390,030	3,903
Tempo médio por consulta	5,5172		3,533	

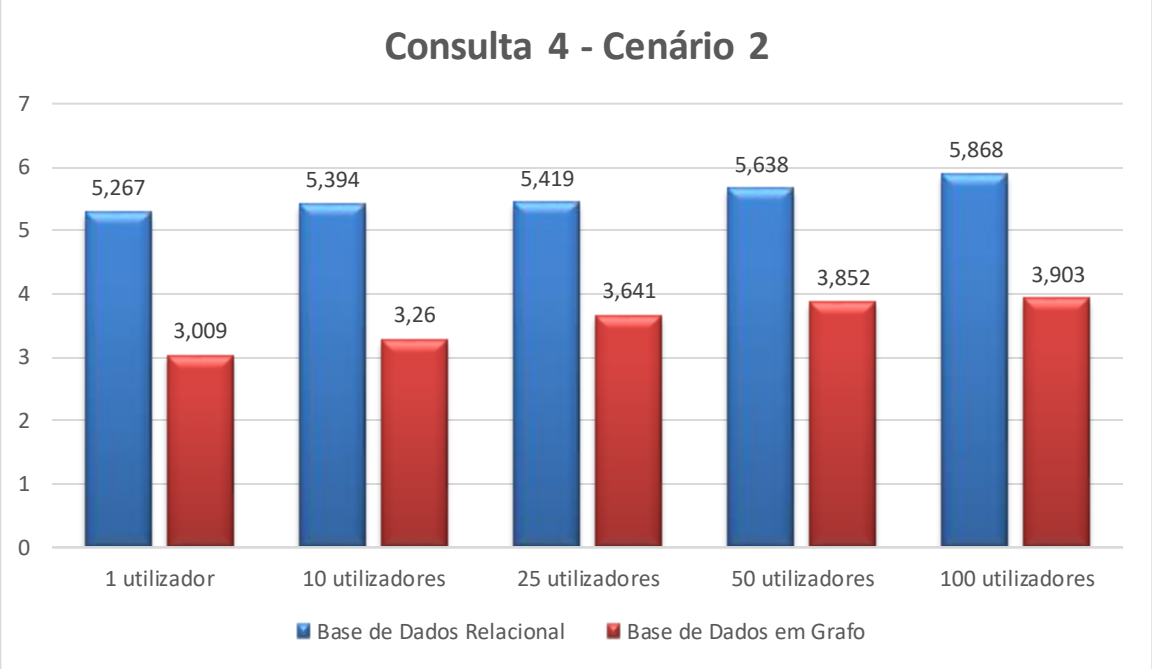

Figura 69 - Gráfico dos resultados da Consulta 4 - Cenário 2.

Tabela 45 - Resultados Consulta 4 - Cenário 3.

Cenário 3	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	9,101	9,101	5,0954	5,095
10 utilizadores	92,018	9,201	53,821	5,382
25 utilizadores	238,615	9,544	136,365	5,454
50 utilizadores	504,357	10,087	262,594	5,651
100 utilizadores	1138,567	11,385	597,187	5,971
Tempo médio por consulta	9,8636		5,5106	

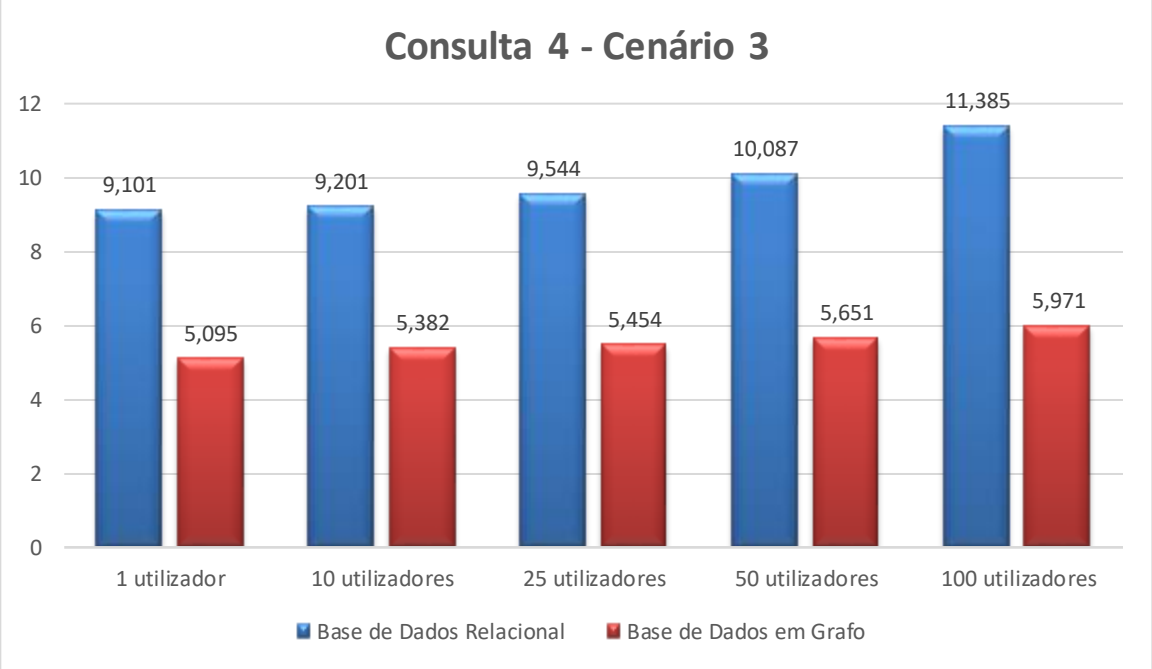

Figura 70 - Gráfico dos resultados da Consulta 4 - Cenário 3.

Tabela 46 - Resultados Consulta 4 - Cenário 4.

Cenário 4	Base de Dados Relacional		Base de Dados em Grafo	
	Tempo Total (s)	Tempo médio de Consulta (s)	Tempo Total (s)	Tempo médio de Consulta (s)
1 utilizador	11,676	11,676	8,050	8,050
10 utilizadores	120,73	12,073	81,645	8,164
25 utilizadores	308,860	12,354	205,271	8,210
50 utilizadores	648,490	12,969	412,046	8,240
100 utilizadores	1344,34	13,443	836,116	8,361
Tempo médio por consulta	12,503		8,205	

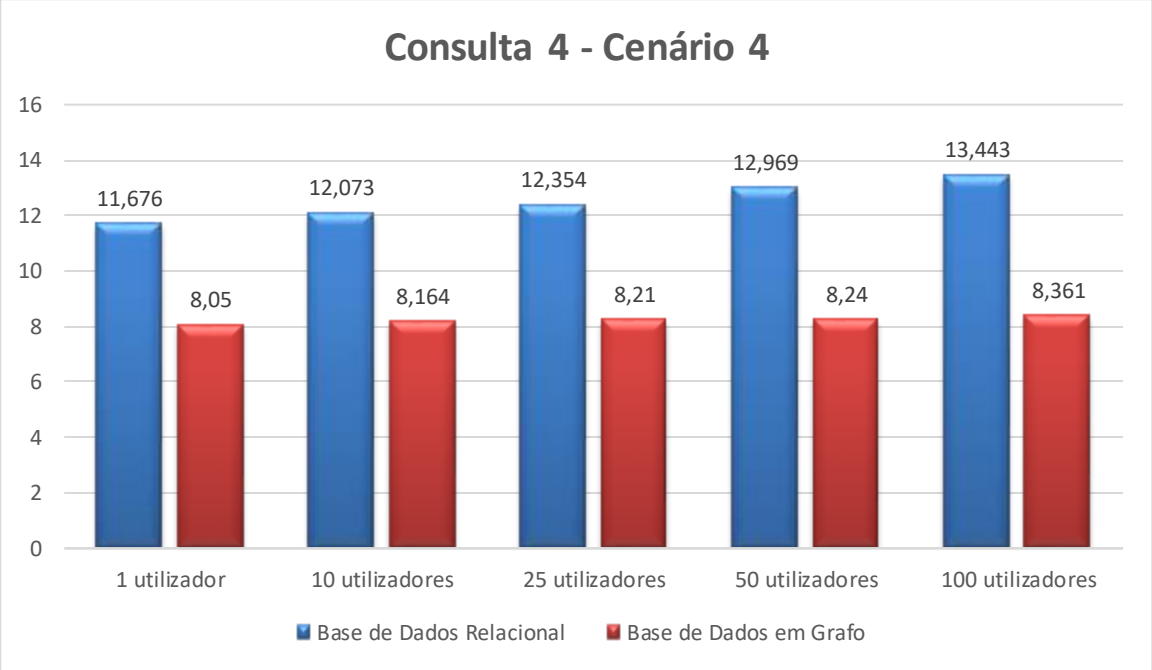

Figura 71 - Gráfico dos resultados da Consulta 4 - Cenário 4.

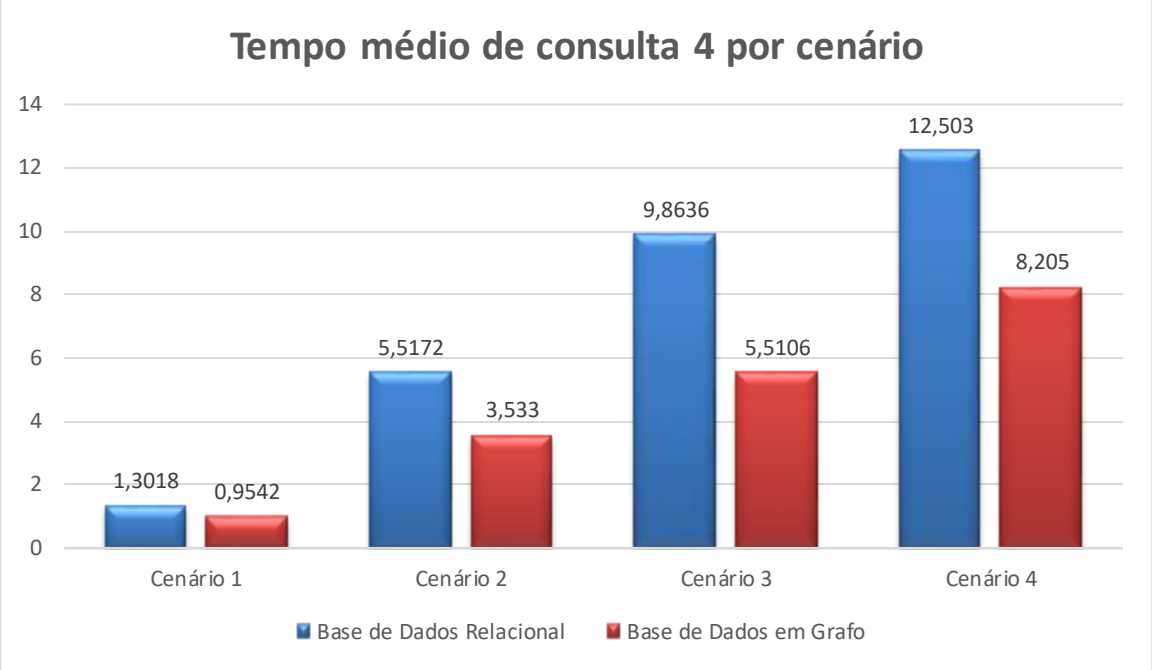

Figura 72 - Tempo médio da consulta 4 por cenário.

Esta consulta 4, demonstra bem a superioridade da base de dados em grafos, mesmo perante consultas que envolvam junções na base de dados em grafo, uma vez que desempenhou melhor em todos os cenários perante diferentes utilizadores. O tempo médio por consulta foi sempre bastante inferior nas consultas às bases de dados em grafos, sendo que a taxa de crescimento por cada cenário é sempre melhor que nas bases de dados relacionais, que tem um crescimento exponencial com grau mais elevado. Um dos objetivos desta consulta era o de verificar qual o desempenho dos dois sistemas de base de dados perante uma consulta que devolvesse uma enorme quantidade de atributos, e a base de dados em grafo desempenhou sempre melhor que a base de dados relacional. O resultado das consultas 3 e 4 vêm comprovar alguns testes realizados, onde as bases de dados em grafos desempenharam melhor que a base de dados relacional na presença de *queries* que envolviam "*joins*" (Modelo relacional) ou relações entre os dados (Modelo em grafo) (Vicknair et al., 2010).

6. CONCLUSÃO

Neste capítulo são apresentadas as conclusões do trabalho realizado. Está dividido em três secções, sendo que no primeiro ponto são apresentadas todas as contribuições que este projeto de dissertação pode dar. Na segunda secção são apresentadas todas as dificuldades e limitações que foram surgindo no decorrer do projeto. Por fim, na terceira e última secção é apresentado o trabalho futuro a ser realizado.

6.1 Contribuições

Hoje em dia, torna-se cada vez mais importante tirar proveito da análise de toda a informação que todos os dias é "despejada" na *Web*. Como o modelo relacional está a apresentar dificuldades em armazenar e analisar estas quantidades enormes de dados, surgiu uma nova gama de bases de dados a que se dá o nome de *NoSQL*. Estas são cada vez mais uma alternativa viável principalmente quando se quer disponibilidade e tolerância ao particionamento, facto que a base de dados relacional não possui. Durante a fase inicial do projeto foram consultadas diversas fontes de modo a conseguir obter uma base teórica que permitisse ter uma boa visão de temas que despertam cada vez mais a curiosidade de todos, entre os quais, todas as vantagens inerentes à análise da enorme quantidade de dados que tem vindo a surgir de forma exponencial na internet. Para armazenar estes dados, surgiram 4 novos tipos de bases de dados, uma vez que a base de dados relacional começou a apresentar problemas no seu armazenamento. Estes novos quatros modelos de base de dados estão divididos em: orientados a Documentos, pares Chave-valor, orientado a Colunas e orientado a Grafos.

No que toca às bases de dados em grafo, estas são indicadas para armazenar dados que se relacionam entre si. As redes sociais, a semântica *web*, o armazenamento de dados químicos e biológicos são as principais aplicações desta gama de base de dados, uma vez que permite descobrir padrões, devolver o caminho mais perto entre dois nodos, entre muitas outras características que nenhuma outra gama de base de dados dispõe.

Neste que toca à preparação dos dados, dos cenários e das consultas, foi possível verificar que os dados não se encontravam nas melhores condições para serem testados. Graças à preparação dos dados foi possível criar um modelo em grafo que correspondeu positivamente aos testes a ele efetuados. Foram efetuados testes na linguagem *Cypher* e *SQL*, que permitiram, sobretudo na linguagem *Cypher* que

desconhecíamos, uma maior familiaridade e conhecimento de como esta funciona. Como foi possível verificar nos testes de carga realizados, o *Neo4J*, de uma maneira geral desempenhou bastante melhor quando as consultas envolviam relações entre dados. O seu desempenho comparativamente com a base de dados relacional alojada no *SQL Server 2014*, e de uma maneira geral, vai melhorando à medida que a quantidade de dados aumenta. Estes resultados vieram comprovar os testes realizados por alguns autores que referem que as bases de dados em grafos obtêm melhores resultados perante consultas que envolvem relações entre dados, facto comprovado nos testes realizados à consulta 2 e 3 que envolviam bastantes *joins* ao modelo relacional. Outro facto que importa realçar é que a base de dados em grafo desempenhou melhor quando o principal objetivo era verificar o comportamento das consultas quando se devolve uma variedade bastante grande de atributos, facto comprovado através da consulta 4. A consulta 1 ajudou a concluir que a base de dados relacional desempenha bastante melhor quando se recorre a funções predefinidas, como por exemplo o *AVG* para fazer uma média.

De uma maneira global, os objetivos para este projeto de dissertação foram alcançados com sucesso, como se pôde demonstrar ao longo do relatório. De realçar ainda, a participação com um poster na conferência do *CAPSI* 2017 – Conferência da Associação Portuguesa de Sistemas de Informação, realizado na Universidade do Minho, na cidade de Guimarães durante os dias 6 e 7 de junho de 2017.

6.2 Dificuldades e limitações

Como em todos os projetos há limitações, este não foge à regra. A principal limitação foi o facto de não conseguir simular os testes de carga na ferramenta *Apache JMeter*. Depois de muitas tentativas, de muito código em *Java*, não foi possível conectar o código *Java* à base de dados em grafo.

Outra limitação passa pelo facto de os testes serem realizados apenas num só conjunto de dados com as suas características próprias. Se fossem realizados mais testes com outro tipo de dados, os resultados poderiam ser diferentes, em benefício ou não da base de dados em grafo.

Outra limitação que não é deste projeto de dissertação, mas sim do produto *Neo4j* é o facto de as transações serem do tipo *ACID* e não seguir o principio *BASE* (*Basic Availability*, *Soft-State* e *Eventual Consistency*). Ou seja, o *Neo4j* fornece um serviço *CA*, isto é, um serviço consistente e disponível, não sendo tolerante a partições. Como as bases de dados *NoSQL* têm a tendência de seguirem o principio *BASE*, e o *Neo4J* não, esta é uma das principais limitações deste produto, e consecutivamente do trabalho realizado.

6.3 Trabalho futuro

No desenvolvimento desta dissertação vários caminhos podiam ter sido tomados, os quais identificam possibilidades que ficaram por investigar.

O facto de as consultas serem submetidas a um software de medição de desempenho como o *Apache JMeter* será um ponto positivo no trabalho, assim as consultas simultâneas com diversos utilizadores serão mais realísticas.

Um aspeto que se pode melhorar é realizar outro tipo de consultas, nomeadamente com *Updates* e *Deletes*, assim como criar diferentes consultas de *Read*, além das quatro já realizadas. Utilizar outro conjunto de dados seria também benéfico para avaliar os dois sistemas de base de dados, uma vez que neste projeto estamos restritos a dados com as mesmas características (aviões).

BIBLIOGRAFIA.

Abramova, V., Bernardino, J., & Furtado, P. (2014). Experimental Evaluation of Nosql Databases. In *International Journal of Database Management Systems (IJDMS) Vol.6, No.3, June 2014* (Vol. 6, pp. 1–16). https://doi.org/10.5121/ijdms.2014.6301

Angles, R. (2012, April). A comparison of current graph database models. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on* (pp. 171-177). IEEE. https://doi.org/10.1109/ICDEW.2012.31

Atikoglu, B., State, W. (2012). Workload analysis of a large-scale key-value store. In *ACM SIGMETRICS Performance Evaluation Review* (Vol. 40, No. 1, pp. 53-64). ACM.

Barmpis, K., Kolovos, D. S. (2014). Evaluation of contemporary graph databases for eficient persistence of large-scale models. *Journal of Object Technology*, *13*(3), 1–26. https://doi.org/10.5381/jot.2014.13.3.a3 ´

Braun, W., Murdoch, J. (2007). A first Course in Statistical Programming with R. Cambridge Press University. Retrieved from: http://site.iugaza.edu.ps/biqelan/files/2010/09/Braun-W.J.-Murdoch-D.J.-A-First-Course-in-Statistical-Programming-with-R-CUP-2007ISBN-0521872650175s.pdf

Brewer, E. (2000). Towards Robust Distributed Systems. In *ACM - Symposium on Principles of Distributed Computing* (Vol. 19, pp. 1–12). https://doi.org/10.1145/343477.343502

Brewer, E. (2012). CAP twelve years later: How the "rules" have changed. *IEEE Computer Society*, *45*(2), 23–29. https://doi.org/10.1109/MC.2012.37

Brown, B., Bughin, J., Byers, A. H., Chui, M., Dobbs, R., Manyika, J., & Roxburgh, C. (2011). Big data: the next frontier for innovation, competition, and productivity. *McKinsey Global Institute*. https://doi.org/10.1080/01443610903114527

Bryant, R., Katz, R., & Lazowska, E. (2008). Big-Data Computing: Creating Revolutionary Breakthroughs in Commerce, Science and Society. *Computing Research Association*, 1–15. Retrieved from http://www.just.edu.jo/~amerb/teaching/2-12-13/cs728/20123173012.pdf

Buerli, M., & Obispo, C. (2012). The Current State of Graph Databases. *The current state of graph databases. Department of Computer Science, Cal Poly San Luis Obispo, mbuerli@ calpoly. edu, 32(3), 67-83.* Retrieved from http://www.cs.utexas.edu/users/cannata/dbms/Class Notes/08 Graph_Databases_Survey.pdf

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., ... Gruber, R. E. (2006). Bigtable: A distributed storage system for structured data. *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*, 205–218. https://doi.org/10.1145/1365815.1365816

Chen, C. P., & Zhang, C. Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*, *275*, 314-347.

Couchbase. (2016). Comparing document-oriented and relational data. Accessed in January 21,2017, from http://docs.couchbase.com/developer/dev-guide-3.0/compare-docs-vs-relational.html

Cudré-Mauroux, P., & Elnikety, S. (2011). Graph data management systems for new application domains. *International Conference on Very Large Databases, 4*(12), 1510–1511. Retrieved from http://131.107.65.14/en-us/people/samehe/vldb2012_graph_tutorial.pdf

DBBest Technologies. (2012). Column Oriented Database Technologies. Accessed in January 21, 2017, from https://www.dbbest.com/blog/column-oriented-database-technologies/

DBEngine. (2017). DB Engines Ranking. Accessed in January 15, 2017, from http://db-engines.com/en/ranking

Diana, M. De, & Gerosa, M. A. (2010). Nosql na Web 2.0: Um estudo comparativo de bancos Não-Relacionais para Armazenamento de Dados na Web 2.0. *IX Workshop de Teses E Dissertações Em Banco*

de Dados - WTDBD, 8. Retrieved from http://www.lbd.dcc.ufmg.br/colecoes/wtdbd/2010/sbbd_wtd_12.pdf

Dragolea, L., & Cotîrlea, D. (2009). Benchmarking-a valid strategy for the long term?. *Annales Universitatis Apulensis: Series Oeconomica*, *11*(2), 813. Retrieved from: http://www.oeconomica.uab.ro/upload/lucrari/1120092/23.pdf

Dumbill, E. (2012). What is big data? - O'Reilly Radar. Accessed in December 15, 2016, from http://radar.oreilly.com/2012/01/what-is-big-data.html;

Fowler, M. (2012). Introduction to NoSQL. *GOTO; Conference AARHUS 2012*. Retrieved from http://gotocon.com/aarhus-2012/presentations/show_protected_video.jsp?oid=4031

Gantz, J., & Reinsel, D. (2011). Extracting Value from Chaos State of the Universe : An Executive Summary. *IDC iView*, (June), 1–12. Retrieved from http://idcdocserv.com/1142

Garlasu, D., Sandulescu, V., Halcu, I., Neculoiu, G., Grigoriu, O., Marinescu, M., & Marinescu, V. (2013, January). A big data implementation based on Grid computing. In *Roedunet International Conference (RoEduNet), 2013 11th* (pp. 1-4). IEEE. Retrieved from https://doi.org/10.1109/RoEduNet.2013.6511732

Han, J., Haihong, E., Le, G., & Du, J. (2011, October). Survey on NoSQL database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on* (pp. 363-366). IEEE. https://doi.org/10.1109/ICPCA.2011.6106531

Hernández-Sampieri, R., Fernández-Collado, C., & Baptista-Lucio, P. (2006). *Analisis de los datos cuantitativos. Metodología de la investigación*.

Holzschuher, F., & Peinl, R. (2013, March). Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*(pp. 195-204). ACM. https://doi.org/10.1145/2457317.2457351

Holzschuher, F., & Peinl, R. (2016). Querying a graph database–language selection and performance considerations. *Journal of Computer and System Sciences*, *82*(1), 45-68. https://doi.org/10.1016/j.jcss.2015.06.006

Hunger, M. (2016). From Relation to Graph: A developer's guide. DZone Magazine. Accessed in February 4, 2017, from https://dzone.com/refcardz/from-relational-to-graph-a-developers-guide

JDStraughan. (2013). Introduction to Graph Databases using Neo4j or How I learned to stop worrying and love the graph. Accessed in January 4, 2017, from http://www.jdstraughan.com/2013/07/01/neo4j-or-how-I-learned-to-stop-worrying-and-love-the-graph/

Khan, N., Yaqoob, I., Hashem, I. A. T., Inayat, Z., Mahmoud Ali, W. K., Alam, M., ... & Gani, A. (2014). Big data: survey, technologies, opportunities, and challenges. *The Scientific World Journal*, *2014*. https://doi.org/10.1155/2014/712826

Laney, D. (2001). 3D Data Management: Controlling Data Volume, Velocity, and Variety. *Meta Group Inc*, *96*(2), 510–517. https://doi.org/10.1016/j.infsof.2008.09.005

Leavitt, N. (2010). Will NoSQL Databases Live Up to Their Promise? *Computer*, *43*(2), 12–14. https://doi.org/10.1109/MC.2010.58

Lith, A., & Mattsson, J. (2010). Investigating storage solutions for large data-A comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data. Retrieved from http://publications.lib.chalmers.se/records/fulltext/123839.pdf

Marr, B. (2015). *Big Data: Using SMART big data, analytics and metrics to make better decisions and improve performance*. John Wiley & Sons. https://doi.org/2024194500

Miller, J. J. (2013, March). Graph database applications and concepts with neo4j. In *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA* (Vol. 2324, p. 36).. Retrieved from http://sais.aisnet.org/2013/MillerJ.pdf

Moniruzzaman, A. B. M., & Hossain, S. A. (2013). Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *Nosql Database: New Era of Databases for Big Data Analytics-Classification, Characteristics and Comparison*, *6*(4), 1–14.

Montag, D. (2013). Understanding neo4j scalability. *White Paper, Neotechnology*.

Nayak, A., & Poriya, A. (2013). Type of NOSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems*, *5*(4), 16–19.

Neo4j. (2013). The most important part of Facebook Graph Search is "Graph". Accessed in January 4, 2017, from https://neo4j.com/blog/why-the-most-important-part-of-facebook-graph-search-is-graph/

Neo4j (Sem Data). Accessed in January 17, 2017, from https://neo4j.com/product/

Neubauer, P., & Rodriguez, M. A. (2010). Constructions from Dots and Lines. *Bulletin of the American Society for Information Science and Technology*, *36*(6), 35–41. https://doi.org/10.1002/bult.2010.1720360610

OrientDB. (2017). Accessed in January 16, 2017, from http://orientdb.com/

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems*, *24*(3), 45-77. Retrieved from http://sites.cgu.edu/chatterjees/files/2013/07/jmis-article.pdf

Pritchett, D. (2008). Base: an Acid Alternative. *Queue*, *6*(3), 48–55. https://doi.org/10.1145/1394127.1394128

Reckoning, T. D. (2014). The six most fascinating technology statistics today. Accessed in December 2, 2016, from: http://www.dailyreckoning.com.au/the-six-most-fascinating-technology-statistics-today/2013/06/11/

Richardson. (1999). Pesquisa social: métodos e técnicas. São Paulo: Atlas 3.

Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph Databases: New opportunities for Connected Data*. *Joe Celkos Complete Guide to NoSQL*. https://doi.org/10.1016/B978-0-12-407192-6.00003-0

Rodriguez, M. A., & Neubauer, P. (2011). The Graph Traversal Pattern. In *Graph Data Management: Techniques and Applications* (pp. 1–18).

Roe, C. (2012). ACID vs BASE: The Shifting pH of Database Transaction Processing. Accessed in December 20, 2016, from http://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/

de Oliveira Santos, S., Musicante, M., & Alves, M. H. F. (2016). Linguagens de consulta para bases de dados em grafos: um mapeamento sistemático. *Revista de Informática Teórica e Aplicada*, *23*(1), 10-68.

Santos, & Silva. (2014). Abordagem ao Banco de Dados Orientado a Grafos Neo4j em um Nível Empresarial.

Sartori, A., & Révillion, P. (2001). A Utilização de Pesquisas Exploratórias na Área de Marketing, 21–37.

Shimpi, D. (2013). An overview of Graph Databases. *IJCA Proceedings on International Conference on Recent Trends in Information Technology and Computer Science 2012*, *ICRTITCS*(3), 16–22.

Silvescu, A., Caragea, D., & Atramentov, A. (2010). Graph Databases. *Artificial Intelligence Research Laboratory*, 14. Retrieved from http://www.cse.iitk.ac.in/users/smitr/PhD Resources/Graph Databases.pdf

Simon, H. a. (1997). *The sciences of the artificial, (third edition)*. *Computers & Mathematics with Applications* (Vol. 33). https://doi.org/10.1016/S0898-1221(97)82941-0

Stapenhurst, T. (2009). The Benchmarking Book: A How-to-Guide to Best Practice for Managers and Practioners. Oxford University. Retrived from https://bbu.yolasite.com/resources/benchmarking_book.pdf

Sousa, G. (2015). Document-Based Databases : Estudo Comparativo no Âmbito das Bases de Dados NoSql. In *Dissertação de Mestrado - Escola de Engenharia - Universidade do Minho*.

Sousa, P. (2010). O teorema CAP. Accessed in November 20, 2016, from https://unrealps.wordpress.com/2010/12/28/o-teorema-cap/

Stidston, M. (2014). Business Leaders Need R's not V's: The 5 R's of Big Data | MapR. Accessed in November 19, 2016, from https://www.mapr.com/blog/business-leaders-need-r%E2%80%99s-not-v%E2%80%99s-5-r%E2%80%99s-big-data)

Tauro, C., Aravindh, S., & Shreeharsha, A. (2012). Comparative Study of the New Generation, Agile, Scalable, High Performance NOSQL Databases. *International Journal of Computer Applications*, *48*(20), 1–4. https://doi.org/10.5120/7461-0336

Tesoriero, C. (2013). *Getting started with OrientDB*. *Technology*. Packt Publishing Ltd. https://doi.org/10.1016/S0261-3069(97)88930-3

Tudorica, B. G., & Bucur, C. (2011, June). A comparison between several NoSQL databases with comments and notes. In *Roedunet International Conference (RoEduNet), 2011 10th* (pp. 1-5). IEEE. https://doi.org/10.1109/RoEduNet.2011.5993686

Vaishnavi, V., & Kuechler, B. (2004). Design Science Research in Information Systems Overview of Design Science Research. *Ais*, 45. https://doi.org/10.1007/978-1-4419-5653-8

Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., & Wilkins, D. (2010). A comparison of a graph database and a relational database. *Proceedings of the 48th Annual Southeast Regional Conference on ACM SE 10*, 1. https://doi.org/10.1145/1900008.1900067

Vieira, M. R., Figueiredo, J. M. De, Liberatti, G., & Viebrantz, A. F. M. (2012). Bancos de Dados NoSQL: conceitos, ferramentas, linguagens e estudos de casos no contexto de Big Data. *Simpósio Brasileiro de Bancos de Dados - SBBD 2012*, (1), 1–30. Retrieved from http://data.ime.usp.br/sbbd2012/artigos/pdfs/sbbd_min_01.pdf

Viraj, T. (2016). CAP Theorem. Accessed in December 30, 2016, from http://blingtechs.blogspot.pt/2016/02/cap-theorem.html.

Virgilio, R., Maccioni, A., & Torlone R., (2013). Converting Relational to graph databases. First international Workshop on Graph Data Management Experiences and Systems. Retrieved from https://www.researchgate.net/publication/262253949_Converting_relational_to_graph_databases

Vukotic, A., & Watt, N. (2013). *Neo4j in Action*. (M. P. Co., Ed.). Shelter Island. Retrieved from http://books.google.com/books?id=61GdmgEACAAJ&pgis=1

Waal-Montgomery, M. D. (2016). World's data volume to grow 40% per year & 50times by 2020: Aureus. Accessed in December 29, 2016, from: https://e27.co/worlds-data-volume-to-grow-40-per-year-50-times-by-2020-aureus-20150115-2/

Wang, S., & Wang, H. (2013). a General Structure of Applied Design Research Studies. *Nedsi.Org*. Retrieved from http://www.nedsi.org/proc/2013/proc/p121023011.pdf

Wood, P. T. (2012). Query languages for graph databases. *ACM SIGMOD Record*, *41*(1), 50. https://doi.org/10.1145/2206869.2206879

Yaqoob, I., Hashem, I. A. T., Gani, A., Mokhtar, S., Ahmed, E., Anuar, N. B., & Vasilakos, A. V. (2016). Big data: From beginning to future. *International Journal of Information Management*, *36*(6), 1231–1247. https://doi.org/10.1016/j.ijinfomgt.2016.07.009

ANEXOS

Anexo A – Instalação do *Neo4J*.

O ficheiro de instalação do *Neo4J* pode ser encontrado em: https://neo4j.com/download/.

1 - Fazer *download* da versão "*Community Edition*".

2 - Executar o instalador que acabou de fazer *download*. Nota: O instalador já inclui a versão *JAVA* necessária para executar o *Neo4j*.

3 - Escolher a diretoria e clicar em "*Next*".

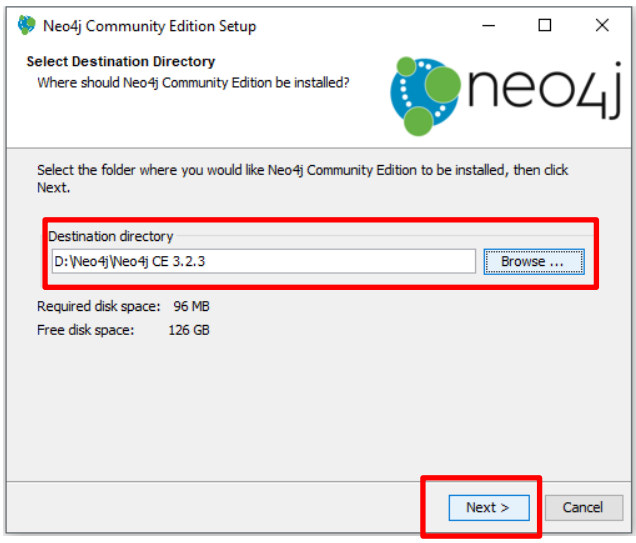

Figura 73 - Diretoria do Neo4j.

4 - Clicar em "*Next*".

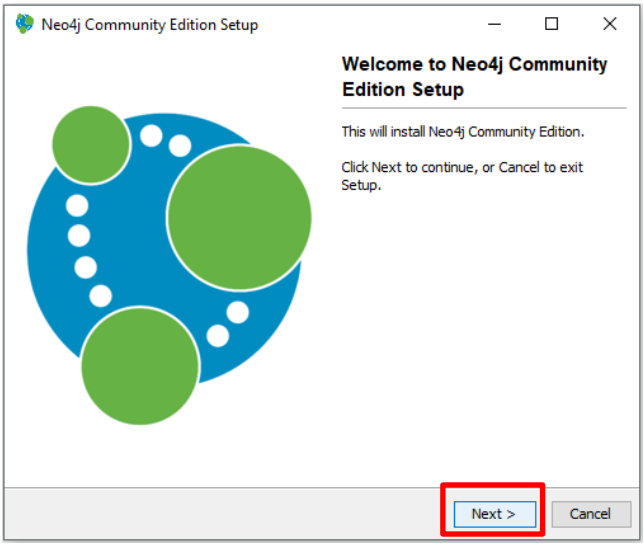

Figura 74 - Instalação Neo4j.

5 - Aceitar os Termos da licença e clicar em "*Next*".

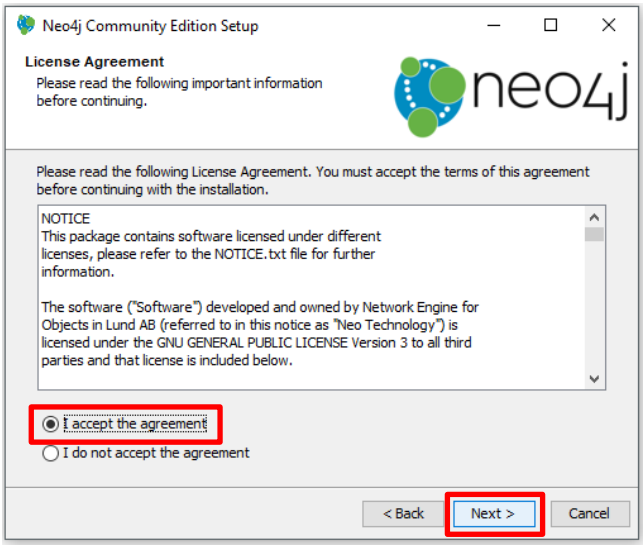

Figura 75 - Termos do Neo4j.

6 - Clicar em "*Next*".

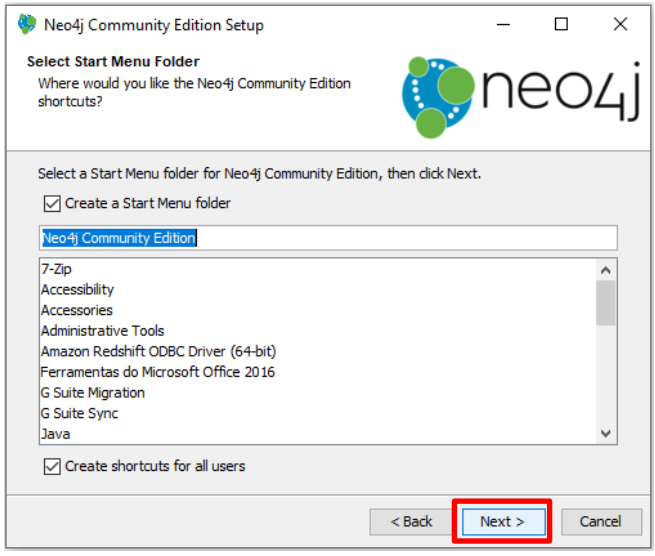

Figura 76 - Nome da pasta Neo4j.

7 - O ficheiro irá ser instalado no computador. Clicar em "*Finish*".

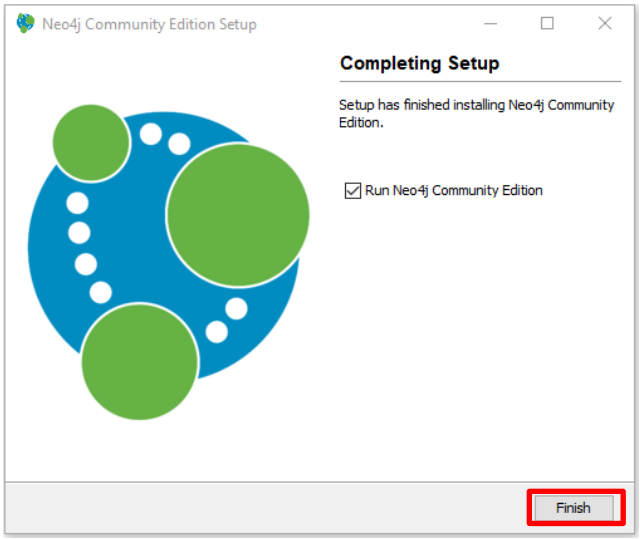

Figura 77 - Instalação completa do Neo4j.

8 - Uma vez instalado, o *Neo4J* irá inicializar. Podemos definir a localização da base de dados, contudo o mais aconselhável é manter a localização por defeito. Clicar em "*Start*".

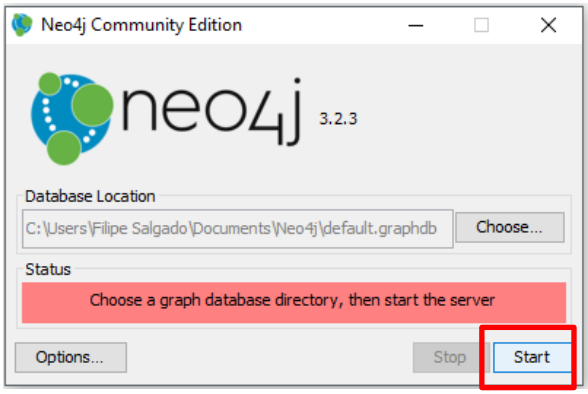

Figura 78 - Iniciar o servidor do Neo4j.

9 - Depois de inicializado o servidor, clicar na ligação para o *Browser*.

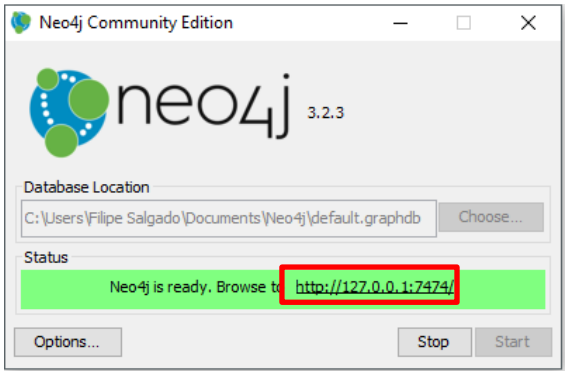

Figura 79 - Hiperligação para aceder ao Neo4j via browser.

10 - Um *browser* irá abrir. Inserir a palavra-passe antiga (por defeito é "*neo4j*"), e a nova palavra-passe. A base de dados está pronta a ser utilizada.

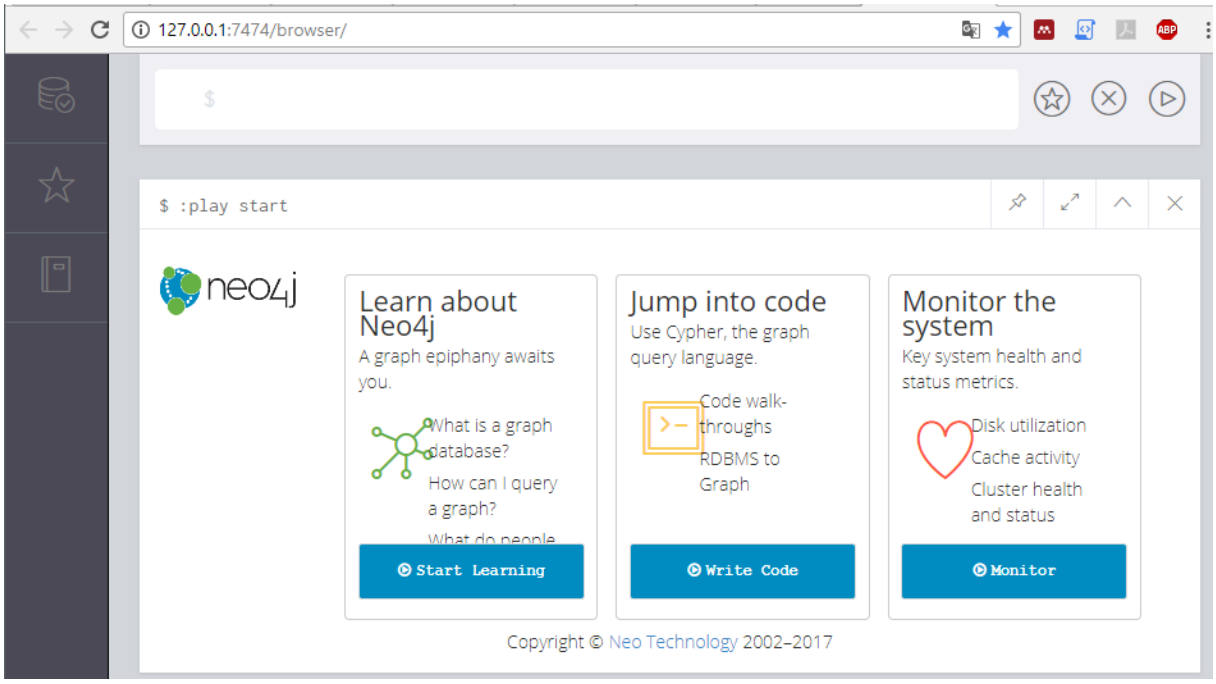

Figura 80 - Interface browser do Neo4j.

Anexo B – Instalação do *SQL Server 2014*.

1 - Para a instalação do *SQL Server 2014* começar por fazer o download da aplicação do website oficial da *Microsoft*.

2 - Selecionar a linguagem pretendida e clicar em "*download*".

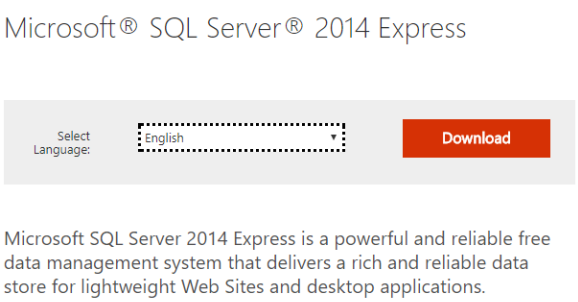

Figura 81 - Linguagem SQL Server.

3 - Escolher a versão compatível com a máquina onde irá instalar o *SQL Server 2014*. A versão *Express* é uma versão livre de custos. Seguidamente clicar em "*Next*". O download iniciar-se-á automaticamente.

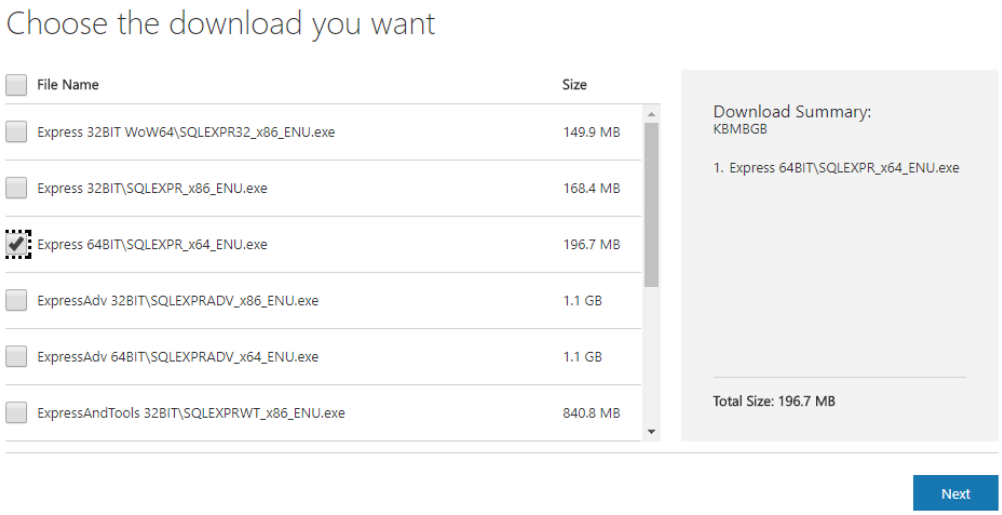

Figura 82 - Escolher versão do SQL Server 2014.

4 - Fazer duplo clique no ficheiro para iniciar a instalação. Clique em "*Ok*" e a extração irá iniciar-se.

5 - Uma vez completa a extração, irá surgir a seguinte janela. Na barra esquerda da janela clicar em "*Installation*" e depois em *"New SQL Server stand-alone installation or add features to na existing installation"*.

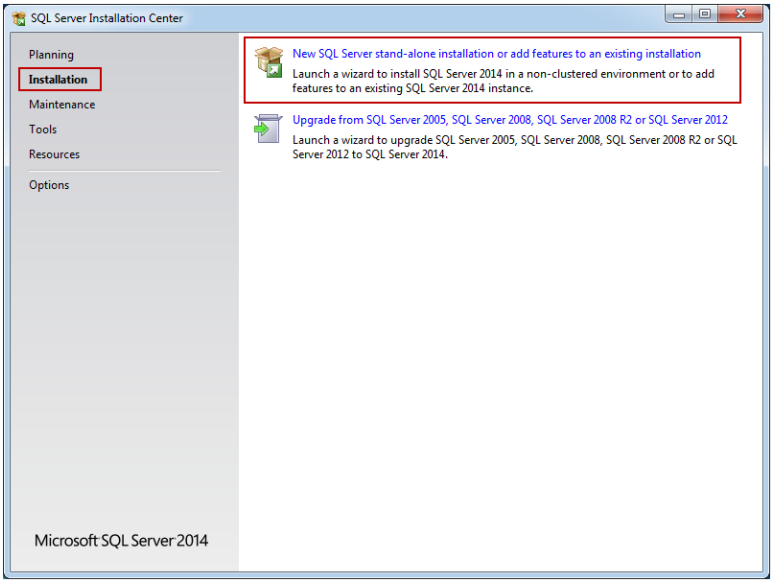

Figura 83 - Nova instalação do SQL Server 2014.

6 - Se todos os pré-requisitos de sistema operativo e da máquina forem cumpridos, a instalação está pronta para ser inicializada. Aceitar os termos da licença do *software*. Assim, deverá selecionar a opção "*I accept the license terms*" e clicar em "*Next*".

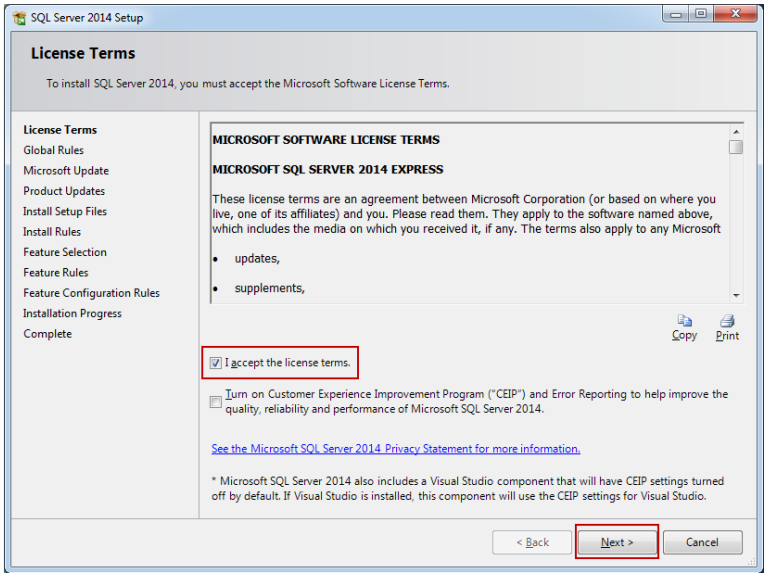

Figura 84 - Aceitar os termos da licença do software.

7 - O próximo passo permite identificar erros que possam ocorrer aquando da instalação do *SQL Server 2014*. Se tudo estiver certo, a instalação irá prosseguir, caso contrário irá ter que as corrigir e reiniciar a instalação.

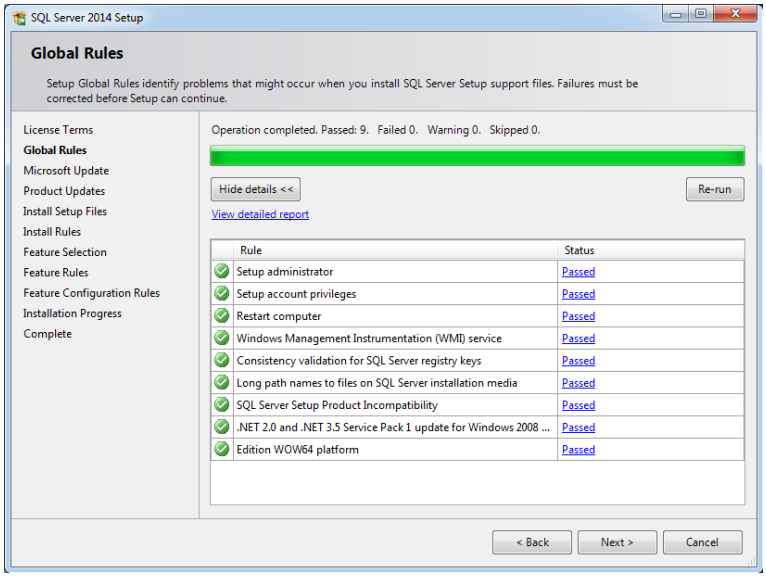

Figura 85 - Global Rules SQL Server 2014.

8 - O passo seguinte irá permitir usar o *Microsoft Update* para verificar a existência de atualizações. É opcional utilizar esta opção. Clicar em "*Next*".

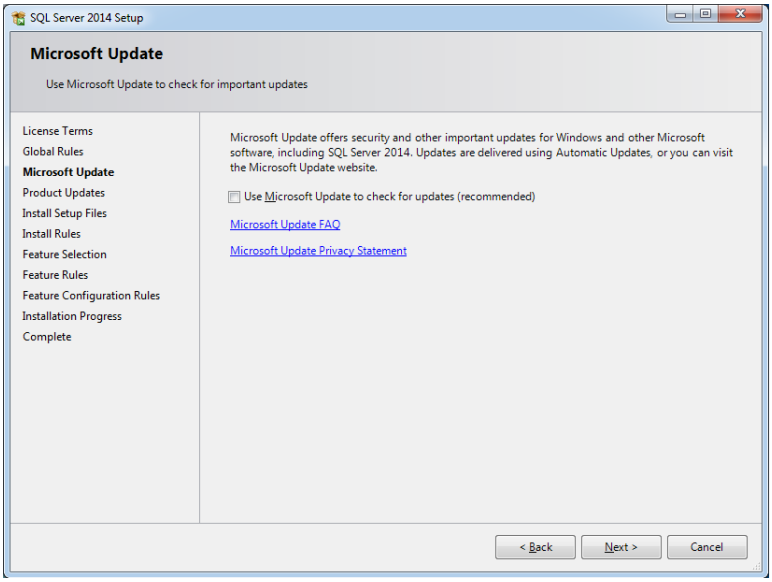

Figura 86 - Instalação do Microsoft Update.

9 - As *Rules* irão ser instaladas automaticamente.

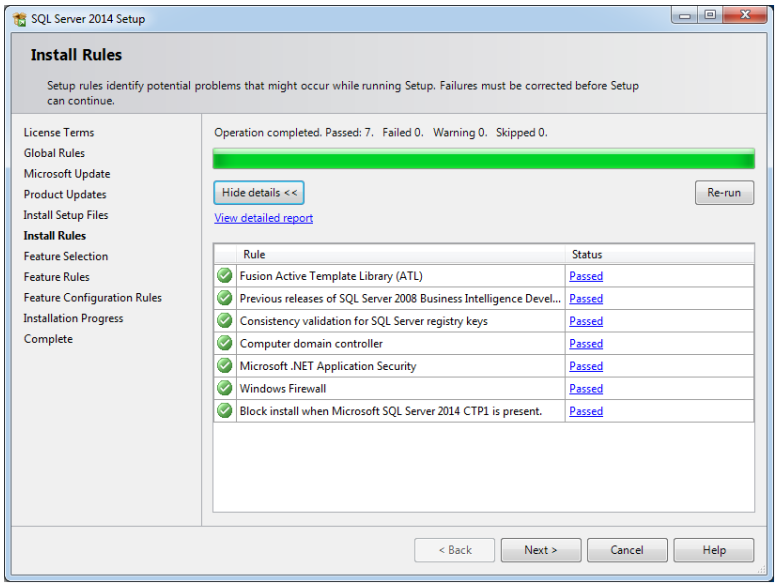

Figura 87 - Instalação das Rules.

10 - Seguidamente selecionar as instâncias a serem instaladas. Selecione "*Management Tools – Basic*" e "*Management Tools – Complete*". Clicar em "*Next*".

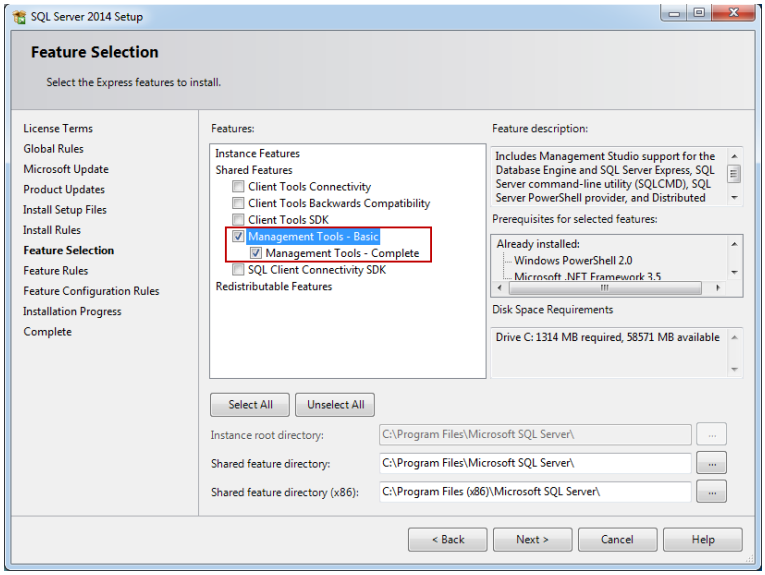

Figura 88 - Instâncias a serem instaladas no SQL Server 2014.

11 - A instalação irá correr e será bem-sucedida.

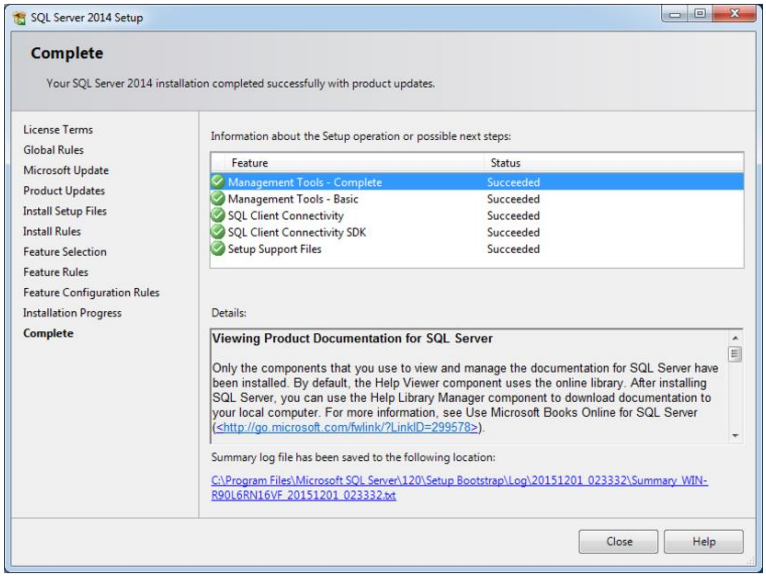

Figura 89 - Instalação completa do SQL Server 2014.

Anexo C – Scripts em *Cypher*.

Criação e Carregamento do nodo *Airplane*.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///airplane.csv" AS row
CREATE (n:Airplane)
SET n = row
```

Criação e Carregamento do nodo *Airport*.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///airport.csv" AS row
CREATE (n:Airport)
SET n = row
```

Criação e Carregamento do nodo *Flight*.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///flights.csv" AS row
CREATE (n:Flight)
SET n = row
```

Criação de índices para os restantes nodos

```
CREATE INDEX ON :Airline(idAirline)
CREATE INDEX ON :Airplane(tailnum)
CREATE INDEX ON :Airport(iata)
CREATE INDEX ON :Flight(idFlight)
```

Criação e Carregamento da relação *By*

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///by.csv" AS row
MATCH (f:Flight),(a:Airline)
WHERE f.idFlight = row.idFlight AND a.idAirline = row.uniqueCarrier
```

```
CREATE (f)-[details:By]->(a)
SET details = row
```

Criação e Carregamento da relação *On*

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///on.csv" AS row
MATCH (f:Flight),(a:Airplane)
WHERE f.idFlight = row.idFlight AND a.tailnum = row.tailNum
CREATE (f)-[details:On]->(a)
SET details = row
```

Criação e Carregamento da relação *From*

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///from.csv" AS row
MATCH (f:Flight),(a:Airport)
WHERE f.idFlight = row.idFlight AND a.iata = row.origin
CREATE (f)-[details:From]->(a)
SET details = row
```

Criação e Carregamento da relação *To*

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///to.csv" AS row
MATCH (f:Flight),(a:Airport)
WHERE f.idFlight = row.idFlight AND a.iata = row.dest
CREATE (f)-[details:To]->(a)
SET details = row
```

Anexo D – Scripts em *SQL*.

Código da criação da base de dados relacional

```
USE [Scn3]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[Aircraft_Manufacturer](
	[idManufacturer] [int] NOT NULL,
	[Manufacturer] [varchar](50) NULL,
 CONSTRAINT [PK_Aircraft_Manufacturer] PRIMARY KEY CLUSTERED
(
	[idManufacturer] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[Aircraft_Model](
	[idAircraft_Model] [int] NOT NULL,
	[aircraft_model] [varchar](50) NULL,
 CONSTRAINT [PK_Aircraft_Model] PRIMARY KEY CLUSTERED
(
	[idAircraft_Model] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
```

```
SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[Aircraft_Type](
	[idAircraft_type] [int] NOT NULL,
	[aircraft_type] [varchar](50) NULL,
 CONSTRAINT [PK_Aircraft_Type] PRIMARY KEY CLUSTERED
(
	[idAircraft_type] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[Airline](
	[idAirline] [varchar](10) NOT NULL,
	[description] [varchar](100) NULL,
 CONSTRAINT [PK_Airline] PRIMARY KEY CLUSTERED
(
	[idAirline] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[Airplane](
	[tailNumber] [varchar](15) NOT NULL,
	[issueData] [varchar](15) NULL,
	[status] [varchar](45) NULL,
	[airplane_year] [int] NULL,
	[idPlane_Type] [int] NULL,
	[idAircraft_type] [int] NULL,
	[idManufacturer] [int] NULL,
	[idAircraft_Model] [int] NULL,
	[idEngine_Type] [int] NULL,
```

```
 CONSTRAINT [PK_Airplane] PRIMARY KEY CLUSTERED
(
	[tailNumber] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[Airport](
	[idAirport] [varchar](10) NOT NULL,
	[airport] [varchar](50) NULL,
	[city] [varchar](50) NULL,
	[state] [varchar](50) NULL,
	[country] [varchar](50) NULL,
 CONSTRAINT [PK_Airport] PRIMARY KEY CLUSTERED
(
	[idAirport] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER
GO

SET ANSI_PADDING ON
GO

GO

SET ANSI_PADDING OFF
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO
```

```
CREATE TABLE [dbo].[Engine_Type](
	[idEngine_Type] [int] NOT NULL,
	[aircraft_name] [varchar](50) NULL,
 CONSTRAINT [PK_Engine_Type] PRIMARY KEY CLUSTERED
(
	[idEngine_Type] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[Flight](
	[flightNum] [int] NULL,
	[year] [int] NULL,
	[month] [int] NULL,
	[dayOfMonth] [int] NULL,
	[dayOfWeek] [int] NULL,
	[actualElapsedTime] [int] NULL,
	[crsElapsedTime] [int] NULL,
	[airTime] [int] NULL,
	[distance] [int] NULL,
	[cancelled] [varchar](50) NULL,
	[idAirportOrigin] [varchar](10) NOT NULL,
	[idAirportDest] [varchar](10) NOT NULL,
	[TailNum] [varchar](15) NOT NULL,
	[idAirline] [varchar](10) NOT NULL,
	[depTime] [int] NULL,
	[crsDepTime] [int] NULL,
	[arrTime] [int] NULL,
	[crsArrTime] [varchar](10) NULL,
	[arrDelay] [int] NULL,
	[depDelay] [int] NULL,
	[taxiIn] [int] NULL,
	[taxiOut] [int] NULL,
	[diverted] [varchar](5) NULL,
	[id_flight] [int] NOT NULL,
 CONSTRAINT [PK_Flight] PRIMARY KEY CLUSTERED
(
	[id_flight] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO
```

```
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[Plane_Type](
	[idPlane_Type] [int] NOT NULL,
	[plane_Type] [varchar](50) NULL,
 CONSTRAINT [PK_Plane_Type] PRIMARY KEY CLUSTERED
(
	[idPlane_Type] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO

ALTER TABLE [dbo].[Airplane]  WITH CHECK ADD  CONSTRAINT
[FK_Airplane_Aircraft_Manufacturer] FOREIGN KEY([idManufacturer])
REFERENCES [dbo].[Aircraft_Manufacturer] ([idManufacturer])
GO

ALTER TABLE [dbo].[Airplane] CHECK CONSTRAINT [FK_Airplane_Aircraft_Manufacturer]
GO

ALTER TABLE [dbo].[Airplane]  WITH CHECK ADD  CONSTRAINT [FK_Airplane_Aircraft_Model]
FOREIGN KEY([idAircraft_Model])
REFERENCES [dbo].[Aircraft_Model] ([idAircraft_Model])
GO

ALTER TABLE [dbo].[Airplane] CHECK CONSTRAINT [FK_Airplane_Aircraft_Model]
GO
ALTER TABLE [dbo].[Airplane]  WITH CHECK ADD  CONSTRAINT [FK_Airplane_Aircraft_Type]
FOREIGN KEY([idAircraft_type])
REFERENCES [dbo].[Aircraft_Type] ([idAircraft_type])
GO

ALTER TABLE [dbo].[Airplane] CHECK CONSTRAINT [FK_Airplane_Aircraft_Type]
GO

ALTER TABLE [dbo].[Airplane]  WITH CHECK ADD  CONSTRAINT [FK_Airplane_Engine_Type]
FOREIGN KEY([idEngine_Type])
REFERENCES [dbo].[Engine_Type] ([idEngine_Type])
GO

ALTER TABLE [dbo].[Airplane] CHECK CONSTRAINT [FK_Airplane_Engine_Type]
GO

ALTER TABLE [dbo].[Airplane]  WITH CHECK ADD  CONSTRAINT [FK_Airplane_Plane_Type]
FOREIGN KEY([idPlane_Type])
REFERENCES [dbo].[Plane_Type] ([idPlane_Type])
GO

ALTER TABLE [dbo].[Airplane] CHECK CONSTRAINT [FK_Airplane_Plane_Type]
GO
```

```
ALTER TABLE [dbo].[Flight]  WITH CHECK ADD  CONSTRAINT [FK_Flight_Airline] FOREIGN
KEY([idAirline])
REFERENCES [dbo].[Airline] ([idAirline])
GO

ALTER TABLE [dbo].[Flight] CHECK CONSTRAINT [FK_Flight_Airline]
GO

ALTER TABLE [dbo].[Flight]  WITH CHECK ADD  CONSTRAINT [FK_Flight_Airplane] FOREIGN
KEY([TailNum])
REFERENCES [dbo].[Airplane] ([tailNumber])
GO

ALTER TABLE [dbo].[Flight] CHECK CONSTRAINT [FK_Flight_Airplane]
GO

ALTER TABLE [dbo].[Flight]  WITH CHECK ADD  CONSTRAINT [FK_Flight_Airport] FOREIGN
KEY([idAirportDest])
REFERENCES [dbo].[Airport] ([idAirport])
GO

ALTER TABLE [dbo].[Flight] CHECK CONSTRAINT [FK_Flight_Airport]
GO

ALTER TABLE [dbo].[Flight]  WITH CHECK ADD  CONSTRAINT [FK_Flight_Airport1] FOREIGN
KEY([idAirportOrigin])
REFERENCES [dbo].[Airport] ([idAirport])
GO

ALTER TABLE [dbo].[Flight] CHECK CONSTRAINT [FK_Flight_Airport1]
GO
```

Carregamento das tabelas da base de dados relacional desde as tabelas temporárias

```
--Carregamento Airline
INSERT INTO [scn1].[dbo].[Airline] (idAirline, [description])
SELECT idAirline, [description]
FROM [Dissertacao].[dbo].[Airline]

--Carregamento Aircraft_type
INSERT INTO [scn1].[dbo].[Aircraft_type]([type])
SELECT DISTINCT [aircraft_type]
FROM [Dissertacao].[dbo].[Airplane]

--Carregamento Engine_type
INSERT INTO [scn1].[dbo].[Engine_type]([type])
SELECT DISTINCT [engine_type]
FROM [Dissertacao].[dbo].[Airplane]

--Carregamento Manufacturer
INSERT INTO [scn1].[dbo].[Manufacturer]([manufacturer])
SELECT DISTINCT [manufacturer]
FROM [Dissertacao].[dbo].[Airplane]

--Carregamento Model
INSERT INTO [scn1].[dbo].[Model]([model])
SELECT DISTINCT [model]
```

```
FROM [Dissertacao].[dbo].[Airplane]


--Carregamento Plane_Type
INSERT INTO [scn1].[dbo].[Plane_Type]([type])
SELECT DISTINCT [plane_type]
FROM [Dissertacao].[dbo].[Airplane]
```

Anexo E – Código da Aplicação de Benchmarking em R.

```
#INSTALL PACKAGES
install.packages("RNeo4j")
install.packages("RODBC")

#LOAD LIBRARIES
library(RNeo4j)
library(RODBC)

#QUERIES IN SQL AND CYPHER. USE DEFINED QUERIES 1, 2, 3 or 4
CypherQuery <- "MATCH (a:Airline {description:\"Delta Air Lines Inc.\"})
RETURN a "
SQLQuery <- "SELECT * FROM Airline WHERE [description] = \'Delta Air
Lines Inc.\'"

#SELECT NUMBER OF USERS (1, 10, 25, 50 or 100)
NumUser <- 1

#SET MAXIMUM PRINT ROWS TO 1000000000
options(max.print=1000000000)

#FUNCTION TO QUERYING AND TEST PERFORMANCE -  NEO4J DATABASE
Neo4jTest <- function(NumUser, CypherQuery){
  start.time <- Sys.time()
  for(i in 1:NumUser){
    graph <- startGraph("http://localhost:7474/db/data/",
username="neo4j", password="qwerty")
    CypherCommand <- cypher(graph, CypherQuery)
    print(CypherCommand)
    print(i)
    end(graph)
  }
  end.time <- Sys.time()
  Cypher.time.taken <- end.time - start.time
  print(paste0("Performance: ", Cypher.time.taken))
}

#FUNCTION TO QUERYING AND TEST PERFORMANCE -  SQL SERVER DATABASE
SQLServerTest <- function(NumUser, SQLQuery){
  start.time <- Sys.time()
  for(i in 1:NumUser){
    con <-odbcConnect("Scn2", uid = "", pwd = "")
    SQLcommand <- RODBC::sqlQuery(con, SQLQuery)
    print(SQLcommand)
    print(i)
    odbcClose(con)
  }
  end.time <- Sys.time()
  SQL.time.taken <- end.time - start.time
  print(paste0("Performance: ", SQL.time.taken))
  }

#RUN SQLServerTest
SQLServerTest(NumUser, SQLQuery)

#RUN Neo4jTest
Neo4jTest(NumUser, CypherQuery)
```

Anexo F – Origem de Dados nas Consultas à Base de Dados Relacional.

Para criar quatro origens de dados *ODBC* diferentes, isto é, uma origem de dados diferente para cada um dos quatro cenários, temos que aceder ao "Administrador da Origem de Dados *ODBC*".

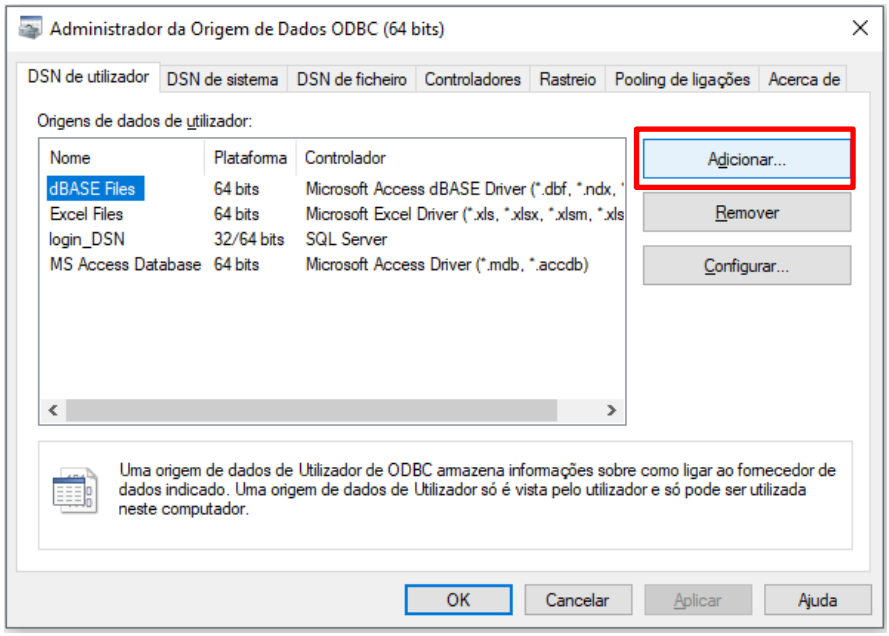

Figura 90 - Administrador da origem de dados ODBC.

Para adicionar uma nova Origem de Dados ODBC, clique em "Adicionar..." e selecione o controlador "*ODBC Driver 11 for SQL Server*", e por fim em "Concluir".

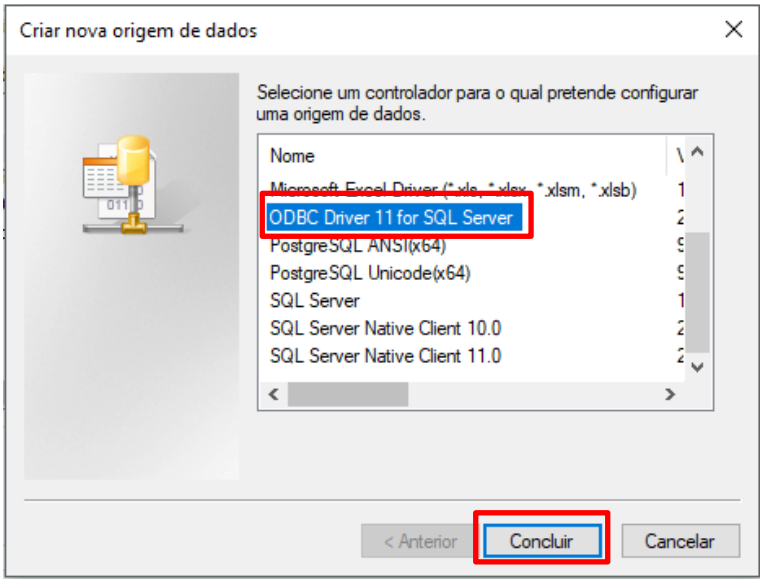

Figura 91 - Criar nova origem de dados.

Comece por dar um nome à sua fonte de dados, uma descrição (opcional) e por fim o nome do servidor onde a fonte de dados se encontra alojado. Clique em "Seguinte". Neste exemplo, o nome dado foi "*Scn1*", por se referir ao Cenário 1.

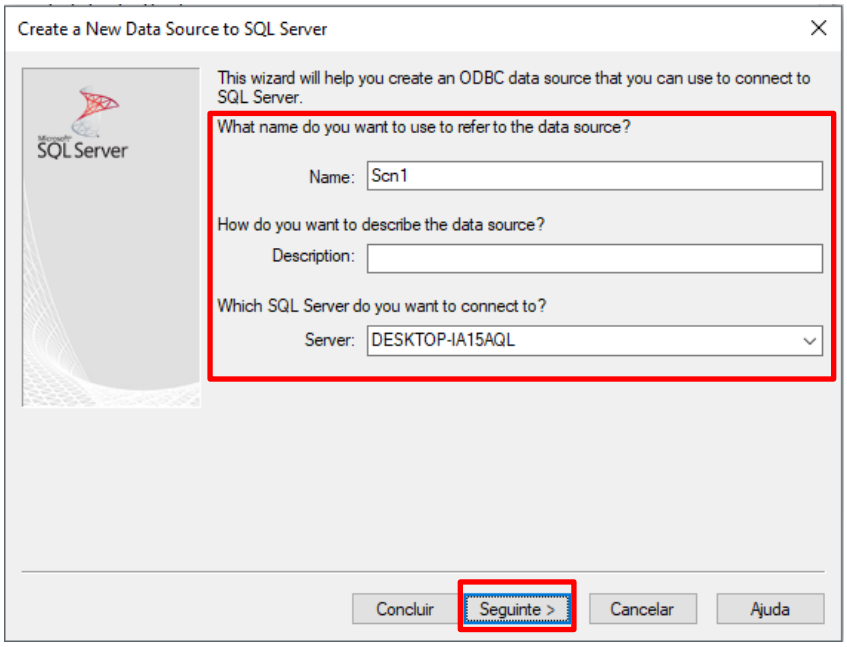

Figura 92 - Nome e servidor da fonte de dados.

Em relação à autenticação da base de dados, no nosso caso, foi utilizada a autenticação integrada do *Windows*. Se preferir, pode utilizar o seu login *ID* e *Password*, se for essa a forma de autenticação no *SQL Server*. Clique em "Seguinte".

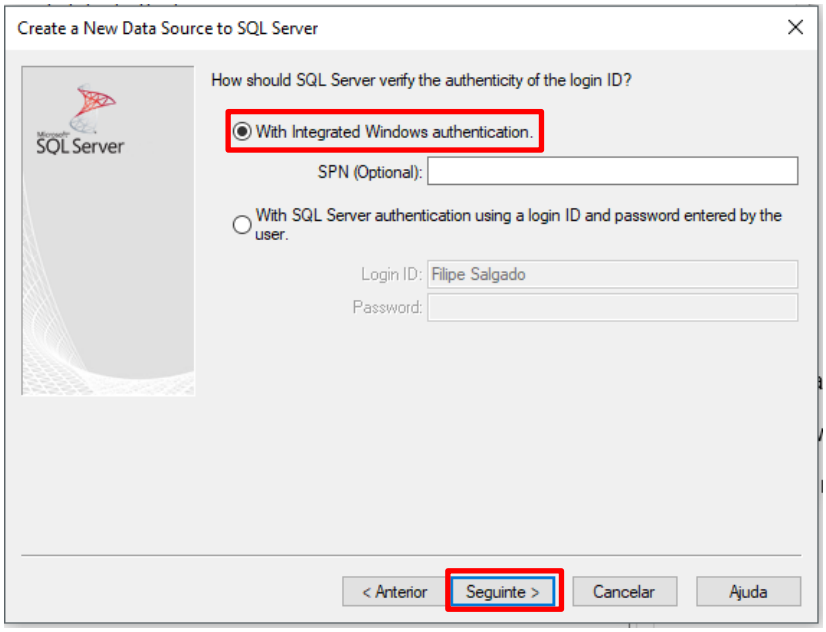

Figura 93 - Tipo de autenticação.

Seguidamente, mude a sua base de dados por defeito para a do cenário que pretende. No nosso caso, selecionamos a do cenário 1 (*Scn1*). Clique em "Seguinte".

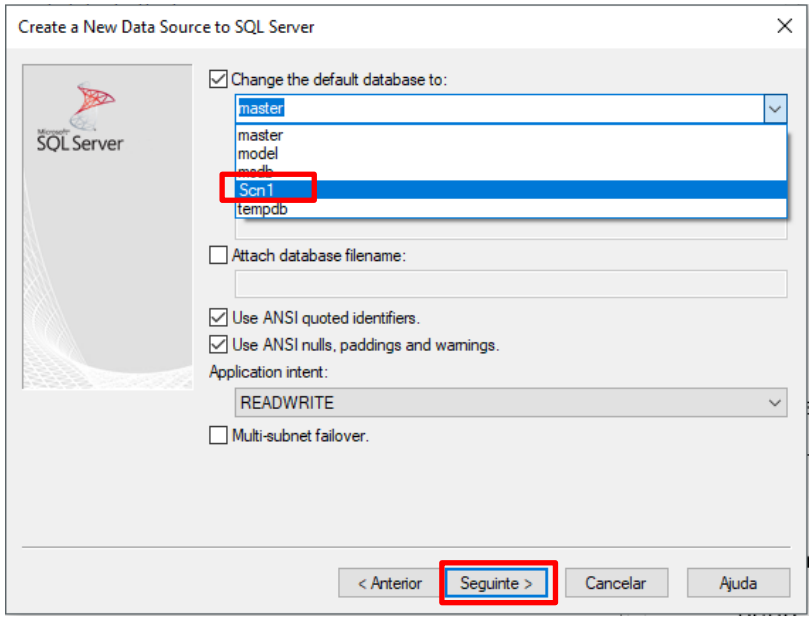

Figura 94 - Base de dados por defeito para a fonte de dados.

Clique em “Concluir”, mantendo as definições padrão.

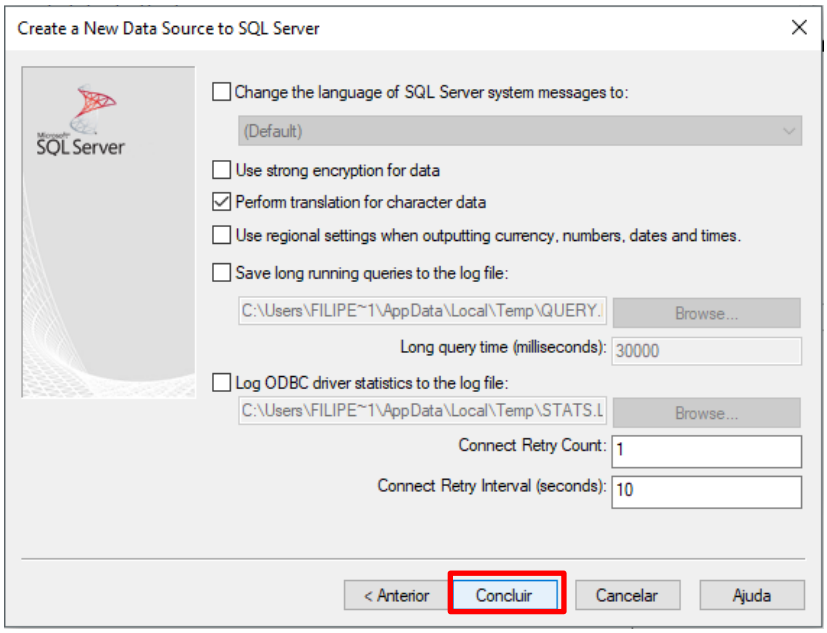

Figura 95 - Definições da nova fonte de dados ODBC.

Clique em “*Test Data Source*”. Caso a sua ligação *ODBC* à base de dados for bem-sucedida, aparecerá a seguinte mensagem:

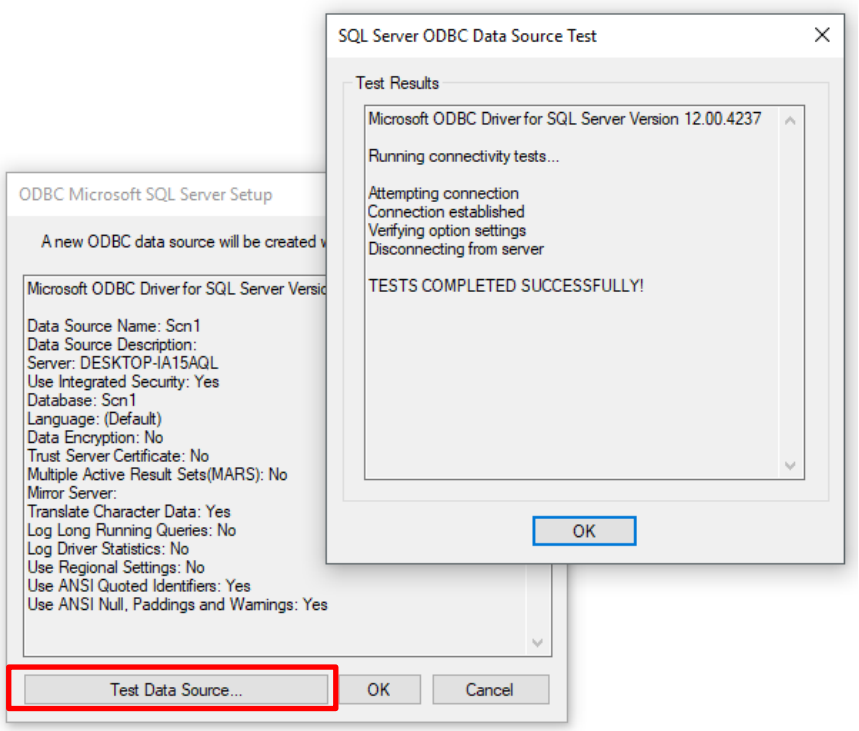

Figura 96 - Testar fonte de dados.

Clique em "*OK*".

Repetindo o mesmo processo para os 4 cenários, teremos então as 4 ligações às Origens de Dados *ODBC*, como se pode comprovar pela seguinte figura.

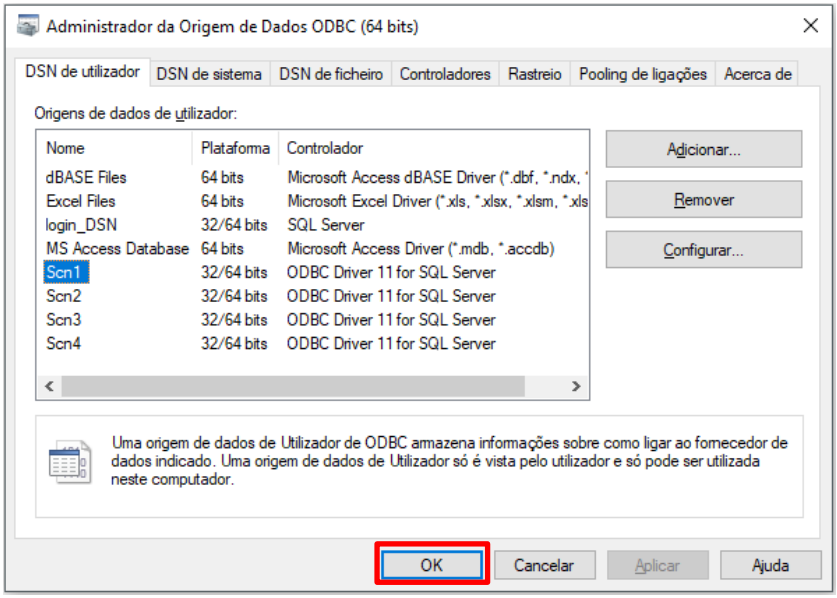

Figura 97 - Origens de dados ODBC.